

05-30-00

A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventorship.....Rosaria et al.
 Applicant.....Microsoft Corporation
 Attorney's Docket No.MS1-556US
 Title: Finite State Model-Based Testing User Interface

TRANSMITTAL LETTER AND CERTIFICATE OF MAILING

To: Commissioner of Patents and Trademarks
 Washington, D.C. 20231

From: David A. Morasch (509) 324-9256
 Lee & Hayes, PLLC
 421 W. Riverside Avenue, Suite 500
 Spokane, WA 99201

The following enumerated items accompany this transmittal letter and are being submitted for the matter identified in the above caption.

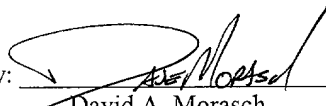
1. Transmittal Letter with Certificate of Mailing included.
2. PTO Return Postcard Receipt
3. Check in the Amount of \$2,254.00
4. Fee Transmittal
5. New patent application (title page plus 63 pages, including claims 1-70 & Abstract)
6. Executed Declaration
7. 10 sheets of formal drawings (Figs. 1-12)
8. Assignment w/Recordation Cover Sheet
9. Information Disclosure Statement, Form PTO-1449 and cited references

Large Entity Status ☒ [x]

Small Entity Status ☐ []

The Commissioner is hereby authorized to charge payment of fees or credit overpayments to Deposit Account No. 12-0769 in connection with any patent application filing fees under 37 CFR 1.16, and any processing fees under 37 CFR 1.17.

Date: 5-26-2000

By: 
 David A. Morasch
 Reg. No. 42,905

CERTIFICATE OF MAILING

I hereby certify that the items listed above as enclosed are being deposited with the U.S. Postal Service as either first class mail, or Express Mail if the blank for Express Mail No. is completed below, in an envelope addressed to The Commissioner of Patents and Trademarks, Washington, D.C. 20231, on the below-indicated date. Any Express Mail No. has also been marked on the listed items.

Express Mail No. (if applicable) EL6 2435 1630

Date: 5/26/2000

By: 
 Lori A. Vierra

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**Finite State Model-Based
Testing User Interface**

Inventor(s):

Steven Rosaria

Henry J. Robinson

ATTORNEY'S DOCKET NO. MS1-556US

TECHNICAL FIELD

The following disclosure relates to finite state model-based testing of systems, and specifically to related user interfaces and methods.

BACKGROUND

Finite state model-based testing describes the behavior of a system (e.g., a software application) under test in terms of a finite state machine. Model-based testing generates software tests from explicit descriptions of an application's behavior. It is desirous to test software applications with a model-based utility because a model of an application is easily updated and is very adaptable to the changing conditions of a software application under test. As a software application evolves, static tests, such as a hard-coded program, have to be modified manually and recompiled whenever there is a change in the specification or functionality of the application. With model-based testing, the changes can be accounted for by simply changing the model. Furthermore, model-based testing enhances the quality of a software application by providing a means for a more accurate and thorough test.

The idea behind model-based testing is to establish a model for the behavior of the application under test. The behavior of the application is described through variables called "operational modes." Operational modes dictate when differing inputs are applicable and how an application will react when the inputs are applied under different operational configurations of the application. For example, whether or not an application is currently running is a common operational mode. Typically, if the application is not running, the only action that

1 a user can execute is to start the application. If the application is running,
2 however, a user typically has a choice of multiple actions that could be executed.

3 A state is a valid combination of values of the operational modes. All of
4 the possible states for an application under test are compiled into a model which is
5 represented by a finite state transition table. Each entry in a state table consists of
6 a current state, an input that is executed, and a next state that describes the
7 condition of the application after the input is applied.

8 A simplified example of a software application from Microsoft Corporation
9 of Redmond, Washington is discussed to illustrate defining a simple finite state
10 model to test the application. Figure 1 shows two possible displays within a Clock
11 window 100 for a "Windows" Clock application: an analog display 102 and a
12 digital display 104. The Clock window also shows a user selectable option,
13 "Settings" 106. Selecting "Settings" initiates a menu (not shown) that enables a
14 user to select either a "Digital" or "Analog" option which changes an analog
15 display to a digital display and vice-versa.

16 In the simplified example of the Clock application, there are four possible
17 system inputs: (1) Start the Clock application; (2) Stop the Clock application;
18 (3) Select Analog setting; and (4) Select Digital setting.

19 There are three rules for each of the four possible system inputs. The rules
20 for each system input are as follows:

21 (1) Start

22 If the application is not running, the user can execute the Start command.

23 If the application is running, the user cannot execute the Start command.

24 After the Start command executes, the application is running.

1 (2) Stop

2 If the application is not running, the user cannot execute the Stop
3 command.

4 If the application is running, the user can execute the Stop command.

5 After the Stop command executes, the application is not running.

6 (3) Analog

7 If the application is not running, the user cannot execute the Analog
8 command.

9 If the application is running, the user can execute the Analog command.

10 After the Analog command executes, the clock is an analog display.

11 (4) Digital

12 If the application is not running, the user cannot execute the Digital
13 command.

14 If the application is running, the user can execute the Digital command.

15 After the Digital command executes, the clock is a digital display.

16
17 The Clock application example will have two operational modes: a system
18 mode and a setting mode, each of which has associated values. The two
19 operational modes and associated values are as follows:

20 System mode: {Running, Not_Running} to indicate whether the Clock
21 application is running or not; and

22 Setting mode: {Analog, Digital} to indicate the format of the clock display.

23
24 From the above Clock application inputs, rules for the application inputs,
25 and the operational modes, a finite state model is defined from all of the possible

state combinations of the application under test. Table 1 below shows the state table for testing the Clock application. The names of the current and next states appear as a combination of operational mode values separated by a period. For example, the state “Running.Digital” means that the Clock application is running and that the display is digital.

<u>Current State</u>	<u>Application Input</u>	<u>Next State</u>
Not_Running.Analog	Start	Running.Analog
Not_Running.Digital	Start	Running.Digital
Running.Analog	Stop	Not_Running.Analog
Running.Digital	Stop	Not_Running.Digital
Running.Analog	Analog	Running.Analog
Running.Analog	Digital	Running.Digital
Running.Digital	Analog	Running.Analog
Running.Digital	Digital	Running.Digital

Table 1

Developing a model is an incremental process that requires simultaneously taking into consideration many aspects of an application under test. State model-based testing is cumbersome and tedious to implement for even a relatively simple software application when attempting to define a model for the application. As the state space increases for a large and complex application (i.e., the increasing number of possible states for an application under development), the model

1 becomes increasingly burdensome to define. This is known as “state explosion”.
2 It is unrealistic to create and maintain a model for an application by hand.

3 Continuing with the example of the Clock application, the model defined
4 by the state table shown in Table 1 above can be graphically represented. Figure 2
5 shows a state transition diagram 200 for the Clock model. The circles 202-208 are
6 nodes that represent the application states and the arrows are links that represent
7 the application inputs 210-224. The direction of the arrows indicate an application
8 input being applied to a current state of the application at the start of an arrow and
9 the resultant next state at the end of the arrow. For example, a “Start” application
10 input 210 applied to the current state “Not_Running.Analog” 202 results in a next
11 state of “Running.Analog” 204.

12 The Clock application model defined by the state table shown above in
13 Table 1 is analyzed by a graph traversal algorithm to develop a path, which is a
14 test sequence, of the application inputs to be applied to the application under test
15 in a particular order. Graph traversal algorithms are stand-alone programs that
16 read in a state transition table from a file and follow a path through the model
17 according to some traversal algorithm. Path segments for a model are graphically
18 represented in the state transition diagram 200 illustrated in Figure 2. Moving
19 from one application state (i.e., nodes 202-208) to another following an
20 application input (i.e., arrow links 210-224) is a path, or a segment of one.

21 Graph traversal algorithms are well known and can be implemented to
22 provide random sequenced paths, minimal duplication paths, or other optimized
23 variations of path sequences. The result of analyzing the Clock application model
24 with a graph traversal algorithm to find an optimal testing sequence is shown in
25 Table 2 below. The testing sequence traverses every application input in the state

model as efficiently as possible. That is to say, the generated test sequence analyzes every application input (i.e., arrow links 210-224) with every associated application state (i.e., nodes 202-208) in the fewest number of link traversals as possible. For clarity, the Table 2 entries are labeled with the corresponding reference numbers from Figure 2.

<u>Test Sequence</u>	<u>Application Input</u>	<u>Next Application State</u>
1	Start (210)	Running.Analog (204)
2	Analog (212)	Running.Analog (204)
3	Digital (214)	Running.Digital (206)
4	Digital (216)	Running.Digital (206)
5	Stop (218)	Not_Running.Digital (208)
6	Start (220)	Running.Digital (206)
7	Analog (222)	Running.Analog (204)
8	Stop (224)	Not_Running.Analog (202)

Table 2

The test sequence generated by a graph traversal algorithm, shown above in Table 2, is applied to a test driver that executes the test sequence on the software application under test. A test driver is a stand-alone program that reads a test sequence from a file (e.g., test_sequence.txt) and initiates testing code that can apply the test sequence to an application under test. The testing code runs the application as if a user was manipulating the application's user-selectable inputs.

“Visual Test” has existing functions that can be called upon by a test driver to implement the testing code for an application test sequence. Table 3 below lists some examples of such functions that would be associated with developing the testing code for the Clock application example.

<u>Functions</u>	<u>Description of Function</u>
Run("C:\WINNT\System32\clock.exe")	Starts the Clock application
WMenuSelect("Settings\Analog")	Selects “Analog” on the “Settings” menu
WSysMenu(0)	Invokes “System” menu for the active window
WFindWnd("Clock")	Finds a window with “Clock” caption
WMenuChecked("Settings\Analog")	Returns “True” if “Analog” is check marked
GetText(0)	Returns the window title of the active window

Table 3

An example of programming code implemented by a test driver to test the first two actions in the test sequence for the Clock application example, “Start” and “Analog”, is shown below. The test execution program would read “Start” from the test_sequence.txt file and execute the Visual Test function Run("C:\WINNT\System32\clock.exe") associated with the “Start” action. The program would then read “Analog” from the file and execute the WMenuSelect("Settings\Analog") function associated with the “Analog” function.

```

1      open "test_sequence.txt" for input as #infile 'get the list of actions
2      while not (EOF(infile))
3          line input #infile, action                'read in an action
4          select case action
5              case "Start"                          'start the Clock
6                  run("C:\WINNT\System32\clock.exe") 'VT call to start Clock program
7              case "Analog"                         'choose analog mode
8                  WMenuSelect("Settings\Analog")    'VT call to select menu item Analog
9              case "Digital"                        'choose digital mode
10                 WMenuSelect("Settings\Digital")    'VT call to select menu item Digital
11             case "Stop"                          'stop the Clock
12                 WSysMenu (0)                      'VT call to bring up system menu
13                 WMenuSelect ("Close")             'VT call to select menu item Close
14             End select
15             Test_oracle()                          'analyze Clock behavior
16         wend

```

After all of the actions in the test program have been executed, a test oracle function (Test_oracle() in the above programming code) can be called to determine if the application behaved as the model expected. A test oracle is a mechanism that verifies if the application has executed correctly. Another one of the benefits of model-based testing is the ability to create a test oracle from the state model. In the case of the simple Clock model example, a test oracle can verify whether the Clock is running or not, and whether the display is in an Analog or Digital mode.

The inventors have developed a system testing interface that provides a user with an easy to use interface to develop and generate a model for a software application under test, initiate the development of a test sequence with a graph traversal algorithm, and initiate the generation of a test execution program to test the application.

SUMMARY

A finite state model-based testing system enables a user to define and generate a model for testing a software application, and to initiate a test of the software application. A system testing interface has a graphical user interface and a model generation engine to implement a model-based testing tool that solves the problem of a user having to define all of the possible states for a software application under test when developing a model for the software application.

The user interface enables a user to define a state table and the associated software application transitions in terms of simple user-defined rules. From the user-defined rules, the model generation engine generates the entire model (i.e., state table) of the software application under test. The state table is generated when the model generation engine calculates and develops all of the possible operational mode value combinations for the software application, and for each permutation, determines which software application inputs are applicable.

After the model of the software application under test is generated, the testing interface enables a user to select a graph traversal program that generates a test sequence of inputs for the software application from the model. The testing interface also enables a user to initiate a test of the application by selecting a test driver program that executes the test sequence of inputs on the software application.

BRIEF DESCRIPTION OF THE DRAWINGS

The same numbers are used throughout the drawings to reference like features and components.

Fig. 1 shows two displays of a software application that is used as an example to illustrate a conventional approach to defining a simple finite state model for the application.

Fig. 2 shows a conventional state transition diagram for the example application shown in Fig. 1.

Fig. 3 is a block diagram that illustrates a finite state model-based testing system.

Fig. 4 is a block diagram of an exemplary computing device that can be used to implement the finite state model-based testing system shown in Fig. 3.

Fig. 5 shows four displays of a software application and is used as an example to illustrate the implementation of the finite state model-based testing system shown in Fig. 3.

Fig. 6 shows a model editor user interface that enables a user to enter software application test information.

Fig. 7 is a block diagram that illustrates multiple linked data structures to store the application test information entered via the user interface shown in Fig. 6.

Fig. 8 shows a rules editor user interface that enables a user to enter software application test information.

Fig. 9 is a block diagram that illustrates multiple linked data structures to store the application test information entered via the user interface shown in Fig. 8.

Fig. 10 shows a graph traversal menu user interface that enables a user to initiate a software application testing component.

Fig. 11 shows a test execution menu user interface that enables a user to initiate a software application testing component.

Fig. 12 is a flow diagram that describes a method to test a software application.

DETAILED DESCRIPTION

Figure 3 illustrates a finite state model-based testing system 300 which enables a user to define and generate a model 302 for testing a software application 304, and to initiate a test of the application 304. Software applications are used in the examples throughout this disclosure to illustrate the finite state model-based testing system 300. Testing system 300 need not be limited to testing software applications, however. A model 302 can be defined and generated with testing system 300 for any computer executable instructions, to include sub-sets of a software application's instructions, any device that can execute computer readable instructions, or any other device that receives user or device generated inputs either directly or remotely, such as via a programmable logic unit hard-wired to a device, or via such technologies as infrared.

Testing system 300 has a system testing interface 306 that enables a user to define a software application under test in terms of simple user-defined rules. The system testing interface 306 is a model-based testing tool that solves the problem of a user having to define all of the possible states for a software application under test when implementing a model 302 for the application.

The system testing interface 306 has a graphical user interface 308 and a model generation engine 310. From the user-defined rules, the model generation engine 310 generates a model 302 of the software application 304. A model of a

1 software application is a state table having multiple state table entries that consist
2 of a current state of the software application, an input of the application, and a
3 next state of the application. The next state of an application indicates the state of
4 the application after the input has been applied to the current state of the
5 application.

6 The graphical user interface 308 has a model editor 312, a rules editor 314,
7 a graph traversal menu 316, and a test execution menu 318, all of which are
8 independently displayable graphical user interfaces. The specifics of each user
9 interface display are described below in conjunction with Figs. 6, 8, 10, and 11,
10 respectively. The model editor 312 and rules editor 314 enable a user to enter
11 parameters to define the model 302 of the software application 304. The graph
12 traversal menu 316 and test execution menu 318 enable a user to initiate a test of
13 the software application 304.

14 The model editor 312 enables a user to enter parameters pertaining to state
15 information about the software application 304. The state information includes
16 operational modes of the application, modal values associated with the operational
17 modes of the application, and inputs to the application. Inputs to a software
18 application can be from a user selection device (e.g., a mouse, a keyboard, a
19 joystick, etc.), or from any other system, device, or application. An “operational
20 mode” is an attribute of a particular state of the software application, and through
21 operational modes, the behavior of the application is described. A “modal value”
22 is associated with a particular operational mode and defines the application
23 behavior described by the operational mode. An application “state” is a valid
24 combination of the modal values of the operational modes of a particular
25

1 application, and the application can have as many states as there are valid
2 combinations of modal values for the particular software application.

3 The rules editor 314 enables a user to enter the parameters pertaining to
4 transition information about the software application 304. The transition
5 information includes a current state of the application which is associated with an
6 input of the application, and a next state of the application. As described above,
7 the next state of an application indicates the state of the application after the input
8 has been applied to the current state of the application. A transition rule is defined
9 by the current state, the input of the application, and the next state.

10 The model generation engine 310 generates the model 302 (i.e., state table)
11 of the software application 304 by determining all of the possible modal value
12 combinations associated with the application operational modes. For each
13 permutation, the model generation engine 310 determines which application inputs
14 are applicable to a current state of the application, where the current state is
15 defined by particular operational modes and associated modal values. In the event
16 that an input can be applied to the current state, the model generation engine 310
17 writes a state table entry out to the state table, where the state table entry includes
18 the current state, the input, and the next state of the software application.

19 The graph traversal menu 316 enables a user to select from among multiple
20 graph traversal algorithms 320(1)-320(N) and to generate a test sequence of
21 application inputs 322 to test the software application 304 in a particular order of
22 application inputs. Graph traversal algorithms are stand-alone programs that read
23 in a state transition table (i.e., a model) from a file and follow a path through the
24 model according to some traversal algorithm. Figure 3 illustrates that the graph
25 traversal menu 316 is the user interface to initiate the graph traversal algorithms

320(1)-320(N). The test sequence of inputs 322 is generated from the model 302 with a particular graph traversal algorithm 320.

The test execution menu 318 enables a user to select from among multiple test driver programs 324(1)-324(M) and to initiate a test of the software application 304. Figure 3 illustrates that the test execution menu 318 is the user interface to initiate the test driver programs 324(1)-324(M). The test driver programs 324(1)-324(M) read in the test sequence of inputs 322 and execute the inputs on the software application under test 304.

Figure 4 illustrates an example of an independent computing device 400 that can be used to implement the components of the finite state model-based testing system 300 (Fig. 3). Computing device 400 includes one or more processors or processing units 402, a system memory 404, and a bus 406 that couples the various system components including the system memory 404 to processors 402. The bus 406 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. The system memory 404 includes read only memory (ROM) 408 and random access memory (RAM) 410. A basic input/output system (BIOS) 412, containing the basic routines that help to transfer information between elements within the computing device 400 is stored in ROM 408.

Computing device 400 further includes a hard drive 414 for reading from and writing to one or more hard disks (not shown). Some computing devices can include a magnetic disk drive 416 for reading from and writing to a removable magnetic disk 418, and an optical disk drive 420 for reading from or writing to a removable optical disk 422 such as a CD ROM or other optical media. The hard

1 drive 414, magnetic disk drive 416, and optical disk drive 420 are connected to the
2 bus 406 by a hard disk drive interface 424, a magnetic disk drive interface 426,
3 and a optical drive interface 428, respectively. Alternatively, the hard drive 414,
4 magnetic disk drive 416, and optical disk drive 420 can be connected to the bus
5 406 by a SCSI interface (not shown).

6 The drives and their associated computer-readable media provide
7 nonvolatile storage of computer-readable instructions, data structures, program
8 modules and other data for computing device 400. Although the exemplary
9 environment described herein employs a hard disk 414, a removable magnetic disk
10 418, and a removable optical disk 422, it should be appreciated by those skilled in
11 the art that other types of computer-readable media which can store data that is
12 accessible by a computer, such as magnetic cassettes, flash memory cards, digital
13 video disks, random access memories (RAMs), read only memories (ROMs), and
14 the like, may also be used in the exemplary operating environment.

15 A number of program modules may be stored on ROM 408, RAM 410, the
16 hard disk 414, magnetic disk 418, or optical disk 422, including an operating
17 system 430, one or more application programs 432, other program modules 434,
18 and program data 436. In some computing devices 400, a user might enter
19 commands and information into the computing device 400 through input devices
20 such as a keyboard 438 and a pointing device 440. Other input devices (not
21 shown) may include a microphone, joystick, game pad, satellite dish, scanner, or
22 the like. In some instances, however, a computing device might not have these
23 types of input devices. These and other input devices are connected to the
24 processing unit 402 through an interface 442 that is coupled to the bus 406. In
25 some computing devices 400, a monitor 444 or other type of display device might

1 also be connected to the bus 406 via an interface, such as a video adapter 446.
2 Some devices, however, do not have these types of display devices. In addition to
3 the monitor 444, computing devices 400 might include other peripheral output
4 devices (not shown) such as speakers and printers.

5 Generally, the data processors of computing device 400 are programmed by
6 means of instructions stored at different times in the various computer-readable
7 storage media of the computer. Programs and operating systems are typically
8 distributed, for example, on floppy disks or CD-ROMs. From there, they are
9 installed or loaded into the secondary memory of a computing device 400. At
10 execution, they are loaded at least partially into the computing device's primary
11 electronic memory. The computing devices described herein include these and
12 other various types of computer-readable storage media when such media contain
13 instructions or programs for implementing the steps described below in
14 conjunction with a microprocessor or other data processor.

15 For purposes of illustration, programs and other executable program
16 components such as the operating system are illustrated herein as discrete blocks,
17 although it is recognized that such programs and components reside at various
18 times in different storage components of the computing device 400, and are
19 executed by the data processor(s) of the computer.

20 Figure 5 shows four displays 500 of a software application that is used as
21 an example to illustrate the implementation of the finite state model-based testing
22 system 300 (Fig. 3). The software application under test is the same "Windows"
23 Clock application described in the "Background" section. In relation to the finite
24 state model-based testing system 300, the software application under test 304 is
25 the Clock application in this example. The example shown in Fig. 5 expands on

1 the simplified example shown in the “Background” section to better illustrate the
2 system testing interface 306 and the complexity of utilizing model-based testing
3 for even small applications.

4 The Clock application displays 500 include a main clock display 502, a font
5 dialog display 504, a clock only display 506, and an “About Clock” dialog display
6 508. The main clock display 502 has a menu 510 that is initiated with a user
7 selectable control, “Settings” 512. The menu 510 enables a user to toggle a
8 seconds, date, and GMT time display. The clock is currently displayed in an
9 analog format, but the display format can be changed via the menu 510 to a digital
10 format (and vice-versa).

11 Selecting “No Title” on the menu 510 displays the clock without a title bar
12 514 as shown in the clock only display 506. A user can toggle back to the main
13 clock display 502 by double clicking a user input 516 at the clock only display
14 506. A user selectable control, “Set Font...”, on the menu 510 initiates the font
15 dialog display 504. The “Set Font...” user selectable control is only activated for
16 selection when the clock is displayed in the digital format. As shown in Fig. 5 on
17 the menu 510, the “Set Font...” user selectable control is not activated because the
18 clock is displayed in the analog format. Another user selectable control, “About
19 Clock...”, on the menu 510 initiates the “About Clock” dialog display 508.

20 Figure 6 shows a graphical user interface display 600 of the model editor
21 312 (Fig. 3) having entry and list fields, as well as user selectable controls, to
22 enable a user to enter the state information to define the model 302 of the software
23 application under test 304. The model editor 312 has an operational modes entry
24 field 602 and an operational modes list field 604. The operational modes fields
25 enable a user to enter operational modes for a software application 304. A user

selectable control 606 adds an operational mode entered in field 602 to the operational modes list in field 604. A user selectable control 608 removes an operational mode from the operational modes list field 604.

The model editor 312 also has a modal values entry field 610 and a modal values list field 612. The modal values fields enable a user to enter modal values associated with the operational modes listed in field 604. As shown in the display 600, the modal values “Main”, “Font”, and “About” in the modal values list field 612 are associated with the operational mode “Window” which is shown selected (hi-lited) in the operational modes list field 604. A user selectable control 614 adds a modal value entered in field 610 to the modal values list in field 612. A user selectable control 616 removes a modal value from the modal values list field 612.

The operational modes and associated modal values for the Clock application example shown in Fig. 5 are shown below in Table 4. The operational modes and modal values are entered via the model editor 312 in the operational modes and modal values fields.

<u>Operational Modes</u>	<u>Modal Values</u>
System	{Not_Invoked, Invoked}
Window	{Main, Font, About}
Setting	{Analog, Digital}
Display	{All, Clock_Only}
WindowSize	{Restored, Maximized, Minimized_from_Maximized, Minimized_from_Restored}

Table 4

The modal values associated with the Clock operational mode “System” indicate whether the Clock application is running or not. The modal values associated with the operational mode “Window” indicate whether the main clock display 502 (Fig. 5), the font dialog display 504, or the “About Clock” dialog display 508 has system focus. The modal values associated with the operational mode “Setting” indicate the Clock display format. The modal values associated with the operational mode “Display” indicate whether the Clock is displayed with a title bar as in the main clock display 502, or without a title bar as in the clock only display 506. The modal values associated with the Clock operational mode “WindowSize” indicate the Clock display window status.

The model editor 312 also has an application input entry field 618 and an application input list field 620. The application inputs fields enable a user to enter inputs of the software application under test. A user selectable control 622 adds software application input entered in field 618 to the application inputs list in field 620. The application inputs for the Clock application example shown in Fig. 5 are listed below in Table 5. The software application inputs are entered via the model editor 312 in the application inputs fields.

<u>Application Input</u>	<u>Description of Input</u>
Analog	Change to analog display.
Digital	Change to digital display.
Set_Font	Set the display font. Available only for digital setting.
GMT	Display the time in GMT.
No_Title	Display without title bar.
Seconds	Toggle the seconds display.

Settings_Date	Toggle the date display.
About	Bring up the About dialog display.
DoubleClick	Toggle between title bar and clock-only display.
Font_OK	Click OK in the Font dialog box.
Font_TypeFont	Type a random font name.
Font_Cancel	Click Cancel in the Font dialog box.
Font_SelectFont	Select a font at random from the font list box.
About_OK	Click OK in the About dialog display.
Invoke	Launch the application.
Terminate_Close	Exit the application by clicking the close window button.
Terminate_Keystroke	Exit the application by pressing Alt-F4.
Maximize	Maximize the application window. Can only be done if the window is not maximized already.
Minimize	Minimize the application window. Can only be done if the window is not minimized already.
Restore_Window	Restore the application window. Restores the window to its original size, which could be either the standard window size or the maximized state.

Table 5

The model editor 312 has a user selectable menu control “Edit” 626 that enables a user to initiate the rules editor 314. A user selectable menu control “Graph” 628 enables a user to initiate the model generation engine 310 to generate the model 302 of a software application under test 304. A user selectable menu control “Tools” 630 enables a user to both initiate the graph traversal menu 316 and the test execution menu 318.

1 The automatic generation of a model (i.e., state table) for a software
2 application under test is independent of the complexity of the application. That is
3 to say, a user is not overly burdened with having to model an increasing number of
4 operational modal values as an application is developed and tested. The system
5 testing interface 306 (Fig. 3) and model generation engine 310 can be utilized to
6 simply recalculate and develop a new model 302 if new operational modal values
7 are needed to test the application.

8 Figure 7 illustrates a data structure 700 having multiple linked data
9 structures. Mode data structures 702(1)-702(5) can be implemented as
10 Component Object Model (COM) objects in a linked list of COM object data
11 structures. The mode data structures 702(1)-702(5) store the operational modes
12 and associated modal values for a software application under test that are entered
13 via the model editor 312 in the operational modes and modal values fields. The
14 mode data structures 702(1)-702(5) in Fig. 7 illustrate the operational modes and
15 associated modal values listed above in Table 4 for the Clock application example
16 shown in Fig. 5. Each mode data structure 702(1)-702(5) stores an operational
17 mode 704(1)-704(5) of the Clock application and the modal values 706(1)(a)-
18 706(5)(d) associated with a particular operational mode. The modal values
19 706(1)(a)-706(5)(d) are also stored in a linked list of one or more modal values
20 associated with an operational mode.

21 Figure 8 shows a user interface display 800 of the rules editor 314 (Fig. 3)
22 having entry and list fields, as well as user selectable controls, to enable a user to
23 enter the transition information to define the model 302 of a software application
24 304. The rules editor 314 has application input entry field 802 to enable a user to
25 enter an input of the software application under test 304. A user selectable control

1 804 initiates a scrollable application inputs list field from which a user can select
2 an input of the application. The application inputs that are listed in the scrollable
3 list field are the software application inputs that are entered via the model editor
4 312 in the application input entry field 618.

5 The rules editor 314 has current state fields 806 to enable a user to enter
6 parameters for a current state of the application. The current state is associated
7 with an application input that can be selected in the application input entry field
8 802. The current state fields 806 include a current state operational mode field
9 808 to enable a user to enter a current state operational mode of the application; a
10 current state modal value field 810 to enable a user to enter modal values
11 associated with the current state operational mode; and a relational operator field
12 812 to enable user selection of a relational operator that indicates the relation of a
13 current state modal value to a current state operational mode. The relational
14 operators that can be selected in the relational operator field 812 are =, ≠, >, ≥, <,
15 and ≤.

16 An operational mode, a modal value, and a relational operator together
17 define a single rule criteria entry. A concatenation operator field 814 enables a
18 user to select a concatenation operator that indicates the relation of a current state
19 rule criteria entry to a second current state rule criteria entry. In this example, the
20 concatenation operators that can be selected in the concatenation operator field
21 814 are either a Boolean AND or a Boolean OR.

22 A current state rule criteria list field 816 shows current state rule criteria
23 entries that have been entered with a user selectable control “Add Criterion” 818.
24 As shown in Fig. 8, the first rule criteria entry in the current state rule criteria list
25 field 816 is “System = Invoked”. The concatenation operator “AND” indicates

1 that "System = Invoked" is ANDed with the second rule criteria entry, "Window =
2 Main", to begin the formation of a current state. The rules editor 312 also has a
3 user selectable control 820 to delete a current state rule criteria entry that has been
4 selected (hi-lited) in the rule criteria list field 816. User selectable control 822
5 clears all entries from the current state fields 806.

6 The parameters to be entered in the current state operational mode field 808
7 and the current state modal value field 810 relate back to the state transition
8 information entered via the model editor 312. A user selectable control 824
9 initiates a scrollable operational mode list from which a user can select an
10 operational mode of the software application. The operational modes that are
11 listed in the scrollable list field are the operational modes that are entered via the
12 model editor 312 in the operational modes entry field 602. A user selectable
13 control 826 initiates a scrollable modal value list from which a user can select a
14 modal value associated with a selected operational mode. The modal values that
15 are listed in the scrollable list field are the modal values that are entered via the
16 model editor 312 in the modal value entry field 610.

17 The rules editor 314 has next state fields 828 to enable a user to enter
18 parameters for a next state of the application. The next state fields 828 include a
19 next state operational mode field 830 to enable a user to enter a next state
20 operational mode of the software application under test; a next state modal value
21 field 832 to enable a user to enter modal values associated with the next state
22 operational mode; and a relational operator 834 to indicate that the relation of a
23 next state modal value to a next state operational mode is "=" (i.e., equal to). Just
24 as with the current state fields 806, an operational mode, a modal value, and a
25 relational operator together define a single rule criteria entry. A concatenation

operator 836 indicates that the relation of a next state rule criteria entry to a second next state rule criteria entry is "AND".

The next state fields 828 also include a numeric value input field 838 and a logical operator input field 840. A logical operator indicates that the next state rule criteria entry is determined with an assignment expression by indicating the relation of the next state rule criteria to a numeric value. The logical operators that can be selected in the logical operator input field 840 are +, -, *, /, and the logical Boolean NOT operator.

A next state rule criteria list field 842 displays next state rule criteria entries that have been entered with a user selectable control "Add Criterion" 844. As shown in Fig. 8, the only rule criteria entry shown in the next state rule criteria list field 842 is "Setting = Analog". The rules editor 314 also has a user selectable control 846 to delete a next state rule criteria entry that has been selected (hi-lited) in the rule criteria list field 842. User selectable control 848 clears all entries from the next state fields 828.

The parameters to be entered in the next state operational mode field 830 and the next state modal value field 832 relate back to the state transition information entered via the model editor 312. A user selectable control 850 initiates a scrollable operational mode list from which a user can select an operational mode of the software application under test. The operational modes that are listed in the scrollable list field are the operational modes that are entered via the model editor 312 in the operational modes entry field 602. A user selectable control 852 initiates a scrollable modal value list from which a user can select a modal value associated with a selected operational mode. The modal

values that are listed in the scrollable list field are the modal values that are entered via the model editor 312 in the modal value entry field 610.

The rules editor 314 has a rules list field 854 to display rule entries that have been entered with a user selectable control “Save Rule” 856. A rule entry in the rules list field 854 includes software application input selected in the application input entry field 802, a current state of the application which is defined by the rule criteria entries in the current state rule criteria list field 816, and a next state of the application which is defined by the rule criteria entries in the next state rule criteria list field 842.

A user selectable control 856 enables a user to disable a rule so that when the model 302 of the software application under test 304 is generated, the model will be defined without the rule. Any disabled rule is shown in the disabled rules list field 858. A user selectable control 860 allows a user to enable a disabled rule so that when the model 302 of the software application 304 is re-generated, the model will again be defined with the rule.

The feature of being able to disable a rule is advantageous when implementing a model-based test. If a particular application input is known to render a developing application under test inoperable, a user can simply disable the particular inoperable input with a single user selection (e.g., a mouse, a keyboard, or other pointing device) and generate a new model to test the application without the input. This is preferable to having to read and modify an entire model to remove all transitions involving the inoperable input, and then consequently having to modify the model again to insert the input back into the model when the application is operating as intended.

1 The rules editor 314 also has a user selectable control 862 to delete a rule
2 entry that has been selected (hi-lited) in the rules list field 854. A user can search
3 the rules listed in the rules list field 854 for a specific application input by
4 selecting the rules search checkbox 864. Selecting the search checkbox 864
5 initiates an application input entry field 866 to enable a user to enter an input of
6 the application to be searched for. A user selectable control 868 initiates a
7 scrollable application inputs list field from which a user can select an input of the
8 software application. The application inputs that are listed in the scrollable list
9 field are the application inputs that are input with the model editor 312 in the
10 application input entry field 618.

11 A complete list of the transition rules for the Clock application shown in
12 Fig. 5 is provided below under the heading "Clock Application Rules". The rules
13 include an application input, a current state or states of the software application,
14 and a next state of the application. A complete model (i.e., state transition table)
15 for the Clock application, as generated by the model generation engine 310, is
16 provided below under the heading "Clock Application Model".

17 Figure 9 illustrates a data structure 900 having multiple linked data
18 structures. Rule data structures 902(1)-902(N) can be implemented as COM
19 objects in a linked list of COM object data structures. The rule data structures
20 902(1)-902(N) store the transition rules from the rules list field 854 (Fig. 8), and
21 each rule data structure 902(1)-902(N) stores the information for one rule. Each
22 rule data structure 902(1)-902(N) has a bit field 904 to indicate whether the rule is
23 enabled or disabled. Additionally, each rule data structure 902(1)-902(N) stores an
24 application input 906, a current state of the application 908, and a next state of the
25 application 910.

1 The current state 908 is stored in a linked list of one or more rule criteria
2 data structures 912(1)-912(3). The next state 910 is also stored in a linked list of
3 one or more rule criteria data structures 914. A rule criteria data structure 912(1)-
4 912(3) for a current state 908 is defined by a RuleCriterion data structure shown
5 following:

```
6 struct RuleCriterion
7 {
8     unsigned short uiOpModelIndex;
9     BYTE bytOperator;
10    unsigned short uiValueIndex;
11    BYTE bytAndOr;
12 };
13
```

14 For a current state of the application 908, a rule criteria data structure
15 912(1)-912(3) stores an operational mode variable (uiOpModelIndex) to indicate an
16 operational mode of the software application under test; a modal value variable
17 (uiValueIndex) to indicate a modal value associated with the operational mode; a
18 relational operator variable (bytOperator) to indicate the relation of the modal value
19 to the operational mode; and a concatenation operator variable (bytAndOr) to
20 indicate the relation of a current state rule criteria entry to a second current state
21 rule criteria entry in the linked list of rule criteria data structures 912(1)-912(3).

22 The variables uiOpModelIndex and uiValueIndex are indices to the respective
23 model editor 312 operational mode and modal values list fields (Fig. 6). The
24 variables assigned to the operational modes shown in the operational modes list
25 field 604, and shown in the modal values list field 612 are as follows:

<u>Operational Mode</u>	<u>uiOpModelIndex</u>	<u>Modal Value</u>	<u>uiValueIndex</u>
System	0		
Window	1	Main	0
		Font	1
		About	2
Setting	2		
Display	3		
WindowSize	4		

The rule criteria data structures 912(1)-912(3) and 914 in Fig. 9 show the values of the RuleCriterion data structure variables for the Clock application input “About” 906. The second current state rule criteria data structure 912(2) shows that the rule criteria entry is “Window = Main”. Accordingly, the uiOpModelIndex = 1 (for Window) and the uiValueIndex = 0 (for Main).

For a next state 910, a rule criteria data structure 914 is defined by a RuleCriterion data structure shown following:

```

struct asgnRuleCriterion
{
    unsigned short uiOpModelIndex;
    BYTE bytOperator;
    unsigned short uiValueIndex;
    BYTE bytAndOr;
    BYTE bytAssignmentType;
    BYTE bytAssignmentOperator;
    double dblAssignmentValue;
};

```

For a next state of the software application 910, a rule criteria data structure 914 stores an operational mode variable (uiOpModelIndex) to indicate an operational mode of the software application under test; a modal value variable (uiValueIndex) to indicate a modal value associated with the operational mode; a

1 relational operator variable (bytOperator) to indicate the relation of the modal value
2 to the operational mode; a concatenation operator variable (bytAndOr) to indicate
3 the relation of a next state rule criteria to a second next state rule criteria in the
4 linked list of rule criteria data structures 914; an assignment variable
5 (bytAssignmentType) to indicate if the next state rule criteria is determined with an
6 assignment expression; a numeric value variable (dblAssignmentValue); and a
7 logical operator variable (bytAssignmentOperator) to indicate the assignment
8 expression relation of the next state rule criteria to the numeric value.

9 The assignment variables bytAssignmentType, bytAssignmentOperator, and
10 dblAssignmentValue are used to define a next state rule criteria of the application.
11 The variables define that the value of a next state rule criteria can be calculated
12 from an assignment expression such as [Current Value] = [Current Value] + 1.

13 Figure 10 shows a graphical user interface display 1000 of the graph
14 traversal menu 316 (Fig. 3) having entry fields that enables a user to select a graph
15 traversal program and generate a test sequence of inputs for the software
16 application under test 304. The graph traversal menu 316 has a graph traversal
17 program field 1002 to enable a user to select the graph traversal program, a model
18 file field 1004 to enable a user to select the model 302 of the software application
19 under test, and a test sequence file field 1006 to enable a user to enter a memory
20 (e.g., disk) storage location for a test sequence file. The test sequence file contains
21 the test sequence of inputs for the software application. The user selectable
22 control "OK" 1008 initiates the generation of the test sequence of application
23 inputs.

24 Figure 11 shows a graphical user interface display 1100 of the test
25 execution menu 318 (Fig. 3) that enables a user to select a test driver program and

1 initiate a test of the software application. The test execution menu 318 has a test
2 driver program field 1102 to enable a user to select a test driver program, a test
3 sequence file field 1104 to enable a user to select a test sequence file containing
4 the test sequence of inputs for the software application under test 304, and a test
5 monitoring interval field 1106 to enable a user to enter a timing interval to define
6 how often the test driver program can be monitored to detect a failure of the test
7 driver program. The user selectable control "OK" 1108 initiates the test of the
8 software application.

9 Figure 12 is a flow diagram that describes a method to test a software
10 application. The method includes presenting a user interface to facilitate entry of
11 state information about a software application to be tested (block 1200). As an
12 example, the user may use the model editor 312 illustrated in Fig. 6 to enter state
13 information. A second user interface may then be presented to facilitate entry of
14 transition information about the software application (block 1202). The rules
15 editor 314 illustrated in Fig. 8 is an example of one such user interface.

16 A model generation engine is initiated to generate a model of the software
17 application from the state and transition information (block 1204). Subsequently,
18 a model is generated at block 1206. The method also includes presenting a user
19 interface to facilitate the selection of a graph traversal program to generate a test
20 sequence of inputs for the software application (block 1208). One example of this
21 user interface is the graph traversal menu 316 illustrated in Fig. 10. When
22 executed, the graph traversal program generates the test sequence of inputs from
23 the model of the software application (illustrated in Fig. 12 as an input of the
24 model from step 1206 to step 1210).

The method further includes presenting a user interface to facilitate the selection of a test driver program to execute the test sequence of inputs on the software application (block 1212). One example of this user interface is the test execution menu 318 illustrated in Fig. 11. When executed, the test driver program executes the test sequence of inputs on the software application under test (the test sequence is illustrated in Fig. 12 as an input to the test driver program from step 1210 to step 1214).

Clock Application Rules

<u>System Input</u>	<u>Current State</u>	<u>New Mode Value(s)</u>
Analog	System = Invoked AND Window = Main AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	Setting = Analog
Digital	System = Invoked AND Window = Main AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	Setting = Digital
Set_Font	System = Invoked AND Window = Main AND Setting = Digital AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Main AND Setting = Digital AND Display = All AND WindowSize = Maximized	Window = Font

1	GMT	System = Invoked AND Window = Main AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	
2			
3	No_Title	System = Invoked AND Window = Main AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	Display = Clock_Only
4			
5	Seconds	System = Invoked AND Window = Main AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	
6			
7	Settings_Date	System = Invoked AND Window = Main AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	
8			
9	About	System = Invoked AND Window = Main AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	Window = About
10			
11	DoubleClick	System = Invoked AND Window = Main AND WindowSize = Restored OR System = Invoked AND Window = Main AND WindowSize = Maximized	Display = Display NOT
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Font_OK	System = Invoked AND Window = Font AND Setting = Digital AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Font AND Setting = Digital AND Display = All AND WindowSize = Maximized	Window = Main
Font_TypeFont	System = Invoked AND Window = Font AND Setting = Digital AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Font AND Setting = Digital AND Display = All AND WindowSize = Maximized	
Font_Cancel	System = Invoked AND Window = Font AND Setting = Digital AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Font AND Setting = Digital AND Display = All AND WindowSize = Maximized	Window = Main
Font_SelectFont	System = Invoked AND Window = Font AND Setting = Digital AND Display = All AND WindowSize = Restored OR System = Invoked AND Window = Font AND Setting = Digital AND Display = All AND WindowSize = Maximized	

1	About_OK	System = Invoked AND Window = About AND Display = All AND WindowSize = Restored OR	Window = Main
2		System = Invoked AND Window = About AND Display = All AND WindowSize = Maximized	
3	Invoke	System = Not_Invoked AND Window = Main AND WindowSize = Restored OR	System = Invoked
4		System = Not_Invoked AND Window = Main AND WindowSize = Maximized	
5	Invoke	System = Not_Invoked AND Window = Main AND Display = All AND WindowSize = Minimized_From_Restored OR	System = Invoked AND WindowSize = Minimized_From_Restored
6		System = Not_Invoked AND Window = Main AND Display = All AND WindowSize = Minimized_From_Maximized	
7	Terminate_Close	System = Invoked AND Window = Main AND Display = All AND WindowSize = Restored OR	System = Not_Invoked
8		System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	

1	Terminate_Keystroke	System = Invoked AND Window = Main AND WindowSize != Minimized_From_Restored AND WindowSize != Minimized_From_Maximized OR System = Invoked AND Window = Main AND Display = All AND WindowSize = Minimized_From_Restored OR System = Invoked AND Window = Main AND Display = All AND WindowSize = Minimized_From_Maximized	System = Not_Invoked
11	Maximize	System = Invoked AND Window = Main AND Display = All AND WindowSize != Maximized	WindowSize = Maximized
13	Minimize	System = Invoked AND Window = Main AND Display = All AND WindowSize = Restored	WindowSize = Minimized_From_Restored
15	Minimize	System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	WindowSize = Minimized_From_Maximized
17	Restore_Window	System = Invoked AND Window = Main AND Display = All AND WindowSize = Minimized_From_Restored	WindowSize = Restored
20	Restore_Window	System = Invoked AND Window = Main AND Display = All AND WindowSize = Minimized_From_Maximized	WindowSize = Maximized
23	Restore_Window	System = Invoked AND Window = Main AND Display = All AND WindowSize = Maximized	WindowSize = Restored

Clock Application Model

<u>Current State</u>	<u>Input</u>	<u>Next State</u>
Not_Invoked Main Analog All Restored	Invoke	Invoked Main Analog All Restored
Not_Invoked Main Analog All Maximized	Invoke	Invoked Main Analog All Maximized
Not_Invoked Main Analog All Minimized_From_Maximized	Invoke	Invoked Main Analog All Minimized_From_Restored
Not_Invoked Main Analog All Minimized_From_Restored	Invoke	Invoked Main Analog All Minimized_From_Restored
Not_Invoked Main Analog Clock_Only Restored	Invoke	Invoked Main Analog Clock_Only Restored
Not_Invoked Main Analog Clock_Only Maximized	Invoke	Invoked Main Analog Clock_Only Maximized
Not_Invoked Main Digital All Restored	Invoke	Invoked Main Digital All Restored
Not_Invoked Main Digital All Maximized	Invoke	Invoked Main Digital All Maximized
Not_Invoked Main Digital All Minimized_From_Maximized	Invoke	Invoked Main Digital All Minimized_From_Restored
Not_Invoked Main Digital All Minimized_From_Restored	Invoke	Invoked Main Digital All Minimized_From_Restored
Not_Invoked Main Digital Clock_Only Restored	Invoke	Invoked Main Digital Clock_Only Restored
Not_Invoked Main Digital Clock_Only Maximized	Invoke	Invoked Main Digital Clock_Only Maximized
Invoked Main Analog All Restored	Analog	Invoked Main Analog All Restored
Invoked Main Analog All Restored	Digital	Invoked Main Digital All Restored
Invoked Main Analog All Restored	GMT	Invoked Main Analog All Restored
Invoked Main Analog All Restored	No_Title	Invoked Main Analog Clock_Only Restored
Invoked Main Analog All Restored	Seconds	Invoked Main Analog All Restored
Invoked Main Analog All Restored	Settings_Date	Invoked Main Analog All Restored
Invoked Main Analog All Restored	About	Invoked About Analog All Restored

1	Invoked Main Analog All Restored	DoubleClick	Invoked Main Analog Clock_Only Restored
2	Invoked Main Analog All Restored	Terminate_Close	Not_Invoked Main Analog All Restored
3	Invoked Main Analog All Restored	Terminate_Keystroke	Not_Invoked Main Analog All Restored
4	Invoked Main Analog All Restored	Maximize	Invoked Main Analog All Maximized
5	Invoked Main Analog All Restored	Minimize	Invoked Main Analog All Minimized_From_Restored
6	Invoked Main Analog All Maximized	Analog	Invoked Main Analog All Maximized
7	Invoked Main Analog All Maximized	Digital	Invoked Main Digital All Maximized
8	Invoked Main Analog All Maximized	GMT	Invoked Main Analog All Maximized
9	Invoked Main Analog All Maximized	No_Title	Invoked Main Analog Clock_Only Maximized
10	Invoked Main Analog All Maximized	Seconds	Invoked Main Analog All Maximized
11	Invoked Main Analog All Maximized	Settings_Date	Invoked Main Analog All Maximized
12	Invoked Main Analog All Maximized	About	Invoked About Analog All Maximized
13	Invoked Main Analog All Maximized	DoubleClick	Invoked Main Analog Clock_Only Maximized
14	Invoked Main Analog All Maximized	Terminate_Close	Not_Invoked Main Analog All Maximized
15	Invoked Main Analog All Maximized	Terminate_Keystroke	Not_Invoked Main Analog All Maximized
16	Invoked Main Analog All Maximized	Minimize	Invoked Main Analog All Minimized_From_Maximized
17	Invoked Main Analog All Maximized	Restore_Window	Invoked Main Analog All Restored
18	Invoked Main Analog All Minimized_From_Maximized	Terminate_Keystroke	Not_Invoked Main Analog All Minimized_From_Maximized
19	Invoked Main Analog All Minimized_From_Maximized	Maximize	Invoked Main Analog All Maximized
20	Invoked Main Analog All Minimized_From_Maximized	Restore_Window	Invoked Main Analog All Maximized
21	Invoked Main Analog All Minimized_From_Restored	Terminate_Keystroke	Not_Invoked Main Analog All Minimized_From_Restored
22	Invoked Main Analog All Minimized_From_Restored	Maximize	Invoked Main Analog All Maximized
23	Invoked Main Analog All Minimized_From_Restored	Maximize	Invoked Main Analog All Maximized
24	Invoked Main Analog All Minimized_From_Restored	Maximize	Invoked Main Analog All Maximized
25	Invoked Main Analog All Minimized_From_Restored	Maximize	Invoked Main Analog All Maximized

1	Invoked Main Analog All Minimized_From_Restored	Restore_Window	Invoked Main Analog All Restored
2	Invoked Main Analog Clock_Only Restored	DoubleClick	Invoked Main Analog All Restored
3	Invoked Main Analog Clock_Only Restored	Terminate_Keystroke	Not_Invoked Main Analog Clock_Only Restored
4	Invoked Main Analog Clock_Only Maximized	DoubleClick	Invoked Main Analog All Maximized
5	Invoked Main Analog Clock_Only Maximized	Terminate_Keystroke	Not_Invoked Main Analog Clock_Only Maximized
6	Invoked Main Digital All Restored	Analog	Invoked Main Analog All Restored
7	Invoked Main Digital All Restored	Digital	Invoked Main Digital All Restored
8	Invoked Main Digital All Restored	Set_Font	Invoked Font Digital All Restored
9	Invoked Main Digital All Restored	GMT	Invoked Main Digital All Restored
10	Invoked Main Digital All Restored	No_Title	Invoked Main Digital Clock_Only Restored
11	Invoked Main Digital All Restored	Seconds	Invoked Main Digital All Restored
12	Invoked Main Digital All Restored	Settings_Date	Invoked Main Digital All Restored
13	Invoked Main Digital All Restored	About	Invoked About Digital All Restored
14	Invoked Main Digital All Restored	DoubleClick	Invoked Main Digital Clock_Only Restored
15	Invoked Main Digital All Restored	Terminate_Close	Not_Invoked Main Digital All Restored
16	Invoked Main Digital All Restored	Terminate_Keystroke	Not_Invoked Main Digital All Restored
17	Invoked Main Digital All Restored	Maximize	Invoked Main Digital All Maximized
18	Invoked Main Digital All Restored	Minimize	Invoked Main Digital All Minimized_From_Restored
19	Invoked Main Digital All Maximized	Analog	Invoked Main Analog All Maximized
20	Invoked Main Digital All Maximized	Digital	Invoked Main Digital All Maximized
21	Invoked Main Digital All Maximized	Set_Font	Invoked Font Digital All Maximized
22	Invoked Main Digital All Maximized	GMT	Invoked Main Digital All Maximized
23	Invoked Main Digital All Maximized	No_Title	Invoked Main Digital Clock_Only Maximized
24	Invoked Main Digital All Maximized		
25	Invoked Main Digital All Maximized		

1	Invoked Main Digital All Maximized	Seconds	Invoked Main Digital All Maximized
2	Invoked Main Digital All Maximized	Settings_Date	Invoked Main Digital All Maximized
3	Invoked Main Digital All Maximized	About	Invoked About Digital All Maximized
4	Invoked Main Digital All Maximized	DoubleClick	Invoked Main Digital Clock_Only Maximized
5	Invoked Main Digital All Maximized	Terminate_Close	Not_Invoked Main Digital All Maximized
6	Invoked Main Digital All Maximized	Terminate_Keystroke	Not_Invoked Main Digital All Maximized
7	Invoked Main Digital All Maximized	Minimize	Invoked Main Digital All Minimized_From Maximized
8	Invoked Main Digital All Maximized	Restore_Window	Invoked Main Digital All Restored
9	Invoked Main Digital All Minimized_From Maximized	Terminate_Keystroke	Not_Invoked Main Digital All Minimized_From Maximized
10	Invoked Main Digital All Minimized_From Maximized	Maximize	Invoked Main Digital All Maximized
11	Invoked Main Digital All Minimized_From Maximized	Restore_Window	Invoked Main Digital All Maximized
12	Invoked Main Digital All Minimized_From Maximized	Terminate_Keystroke	Not_Invoked Main Digital All Minimized_From Restored
13	Invoked Main Digital All Minimized_From Restored	Maximize	Invoked Main Digital All Maximized
14	Invoked Main Digital All Minimized_From Restored	Restore_Window	Invoked Main Digital All Restored
15	Invoked Main Digital Clock_Only Restored	DoubleClick	Invoked Main Digital All Restored
16	Invoked Main Digital Clock_Only Restored	Terminate_Keystroke	Not_Invoked Main Digital Clock_Only Restored
17	Invoked Main Digital Clock_Only Maximized	DoubleClick	Invoked Main Digital All Maximized
18	Invoked Main Digital Clock_Only Maximized	Terminate_Keystroke	Not_Invoked Main Digital Clock_Only Maximized
19	Invoked Font Digital All Restored	Font_OK	Invoked Main Digital All Restored
20	Invoked Font Digital All Restored	Font_TypeFont	Invoked Font Digital All Restored
21	Invoked Font Digital All Restored	Font_Cancel	Invoked Main Digital All Restored
22	Invoked Font Digital All Restored	Font_SelectFont	Invoked Font Digital All Restored
23	Invoked Font Digital All Maximized	Font_OK	Invoked Main Digital All Maximized
24			
25			

1	Invoked Font Digital All Maximized	Font_TypeFont	Invoked Font Digital All Maximized
2	Invoked Font Digital All Maximized	Font_Cancel	Invoked Main Digital All Maximized
3	Invoked Font Digital All Maximized	Font_SelectFont	Invoked Font Digital All Maximized
4	Invoked About Analog All Restored	About_OK	Invoked Main Analog All Restored
5	Invoked About Analog All Maximized	About_OK	Invoked Main Analog All Maximized
6	Invoked About Digital All Restored	About_OK	Invoked Main Digital All Restored
7	Invoked About Digital All Maximized	About_OK	Invoked Main Digital All Maximized

8

9

10 The finite state model-based testing system 300 includes an option that will

11 generate a skeleton of the testing source code for the test drivers for “Microsoft

12 Visual C++”, “Microsoft Visual Basic”, and “Rational Visual Test”. A user can

13 then insert the programmable code for the individual function actions to test a

14 particular software application.

15 The model-based system testing interface 306 minimizes a tester’s burden

16 (e.g., time, labor, and expense) in developing a model 302 for a software

17 application under test, thus facilitating the use of model-based testing. Facilitating

18 the use of model-based testing will result in better test coverage for a particular

19 software application, which in turn will result in applications of higher quality.

20 Although the invention has been described in language specific to structural

21 features and/or methodological steps, it is to be understood that the invention

22 defined in the appended claims is not necessarily limited to the specific features or

23 steps described. Rather, the specific features and steps are disclosed as preferred

24 forms of implementing the claimed invention.

25

CLAIMS

We claim:

1. A finite state model-based testing system comprising:

a model generation engine to generate a model of a software application to be tested; and

a graphical user interface to enable user entry of parameters for defining the model.

2. A finite state model-based testing system as recited in claim 1, wherein the user interface enables a user to enter state information and transition information about the software application, the transition information describing a next state of the software application after an input has been applied to a current state of the software application.

3. A finite state model-based testing system as recited in claim 1, wherein the user interface enables a user to enter state information about the software application, the state information comprising:

an operational mode of the software application, wherein the operational mode is an attribute of a particular state of the software application;

at least one modal value associated with the operational mode, wherein the modal value describes a behavior of the operational mode; and

an input of the software application.

1 **4.** A finite state model-based testing system as recited in claim 1,
2 wherein the user interface enables a user to enter transition information about the
3 software application, the transition information comprising:

4 a current state of the software application, the current state being associated
5 with an input of the software application; and

6 a next state of the software application, the next state indicating the state of
7 the software application after the input has been applied to the current state of the
8 software application.

9
10 **5.** A finite state model-based testing system as recited in claim 1,
11 wherein the user interface comprises a model editor to enable user entry of an
12 operational mode, a modal value associated with the operational mode, and an
13 input of the software application for defining the model.

14
15 **6.** A finite state model-based testing system as recited in claim 1,
16 wherein the user interface comprises a rules editor to enable user entry of an input
17 of the software application, a current state of the software application, and a next
18 state of the software application indicating the state of the software application
19 after the input has been applied to the current state of the software application for
20 defining the model.

1 7. A finite state model-based testing system as recited in claim 1,
2 wherein the model of the software application is a state table, the state table
3 having at least one state table entry, and wherein:

4 a state table entry comprises:

- 5 (1) a current state of the software application;
6 (2) an input of the software application;
7 (3) a next state of the software application, the next state indicating
8 the state of the software application after the input has been applied
9 to the current state of the software application;

10 the model generation engine evaluates the current state of the software
11 application to determine if an input of the software application can be applied to
12 the current state and in the event that the input can be applied to the current state,
13 writes a state table entry out to the state table.

14
15 8. A finite state model-based testing system as recited in claim 1,
16 wherein the user interface comprises a graph traversal menu to enable a user to
17 select a graph traversal program and generate a test sequence of inputs for the
18 software application.

19
20 9. A finite state model-based testing system as recited in claim 1, further
21 comprising a graph traversal program to generate a test sequence of inputs for the
22 software application, the test sequence of inputs generated from the model of the
23 software application with the graph traversal program.

1 **10.** A finite state model-based testing system as recited in claim 1,
2 wherein the user interface comprises a test execution menu to enable a user to
3 select a test driver program and initiate a test of the software application.
4

5 **11.** A finite state model-based testing system as recited in claim 1,
6 further comprising a test driver program to execute a test sequence of application
7 inputs on the software application.
8

9 **12.** A user interface for testing a software application, the user interface
10 comprising:

11 a model editor to enable user entry of state information to define a model of
12 a software application to be tested; and

13 a rules editor to enable user entry of transition information to further define
14 the model of the software application to be tested, the transition information
15 describing a next state of the software application after an input has been applied
16 to a current state of the software application.
17

18 **13.** A user interface as recited in claim 12, wherein the user interface is
19 a graphical user interface.
20
21
22
23
24
25

1 **14.** A user interface as recited in claim 12, wherein the state information
2 comprises:

3 an operational mode of the software application, wherein the operational
4 mode is an attribute of a particular state of the software application;

5 at least one modal value associated with the operational mode, wherein the
6 modal value describes a behavior of the operational mode; and

7 an input of the software application.

8
9 **15.** A user interface as recited in claim 12, wherein the transition
10 information comprises:

11 a current state of the software application, the current state being associated
12 with an input of the software application; and

13 a next state of the software application, the next state indicating the state of
14 the software application after the input has been applied to the current state of the
15 software application.

16
17 **16.** A user interface as recited in claim 12, wherein the model editor
18 comprises:

19 an operational modes entry field to enable user entry of an operational
20 mode of the software application; and

21 an operational modes list field to display the operational mode.

1 **17.** A user interface as recited in claim 12, wherein the model editor
2 comprises:

3 a modal values entry field to enable user entry of a modal value associated
4 with an operational mode of the software application; and
5 a modal values list field to display the modal value.
6

7 **18.** A user interface as recited in claim 12, wherein the model editor
8 comprises:

9 an inputs entry field to enable user entry of an input of the software
10 application; and
11 an inputs list field to display the input of the software application.
12

13 **19.** A user interface as recited in claim 12, wherein the rules editor
14 comprises fields to display the state information that can be entered using the
15 model editor, the fields comprising:

16 inputs fields to display inputs of the software application, wherein the
17 inputs fields also enable user selection of an input of the software application;

18 operational modes fields to display operational modes of the software
19 application, wherein the operational modes fields also enable user selection of an
20 operational mode; and

21 modal values fields to display modal values associated with an operational
22 mode, wherein the modal values fields also enable user selection of a modal value.
23
24
25

1 **20.** A user interface as recited in claim 19, wherein the rules editor
2 enables user entry of the transition information, and wherein:

3 the transition information comprises:

- 4 (1) a current state of the software application, the current state being
5 associated with the input of the software application;
6 (2) a next state of the software application, the next state indicating
7 the state of the software application after the input has been applied
8 to the current state of the software application;

9 the rules editor comprises:

- 10 (1) an inputs field to enable user entry of the input;
11 (2) a current state operational mode field to enable user entry of the
12 operational mode as a current state operational mode;
13 (3) a current state modal value field to enable user entry of the modal
14 value associated with the current state operational mode;
15 (4) a next state operational mode field to enable user entry of the
16 operational mode as a next state operational mode; and
17 (5) a next state modal value field to enable user entry of the modal
18 value associated with the next state operational mode.

1 **21.** A user interface as recited in claim 12, wherein the model is a state
2 table having at least one state table entry, the state table entry having a current
3 state of the software application, an input of the software application, and a next
4 state of the software application, the next state indicating the state of the software
5 application after the input has been applied to the current state of the software
6 application; and

7 the model editor enables user initiation of a model generation engine to
8 generate the model of the software application, the model generation engine being
9 configured to evaluate a current state of the software application to determine if an
10 input of the software application can be applied to the current state and in the
11 event that the input can be applied to the current state, writes a state table entry out
12 to the state table.

13
14 **22.** A user interface as recited in claim 12, wherein the model editor
15 enables user initiation of a graph traversal program to generate a test sequence of
16 inputs for the software application.

17
18 **23.** A user interface as recited in claim 12, wherein the user interface
19 further comprises a graph traversal menu to enable user selection of a graph
20 traversal program to generate a test sequence of inputs for the software
21 application.

1 **24.** A user interface as recited in claim 23, wherein the graph traversal
2 menu comprises:

3 a graph traversal program field to enable user selection of the graph
4 traversal program;

5 a model file field to enable user selection of the model of the software
6 application to be tested; and

7 a test sequence file field to enable user entry of a memory storage location
8 for a test sequence file, the test sequence file containing the test sequence of inputs
9 for the software application.

10
11 **25.** A user interface as recited in claim 12, wherein the model editor
12 enables user initiation of a test driver program to read a test sequence of inputs for
13 the software application and apply the test sequence to the software application.

14
15 **26.** A user interface as recited in claim 12, wherein the user interface
16 further comprises a test execution menu to enable user selection of a test driver
17 program to read a test sequence of inputs for the software application and apply
18 the test sequence to the software application.

19
20 **27.** A user interface as recited in claim 26, wherein the test execution
21 menu comprises:

22 a test driver program field to enable user selection of the test driver
23 program;

24 a test sequence file field to enable user selection of a test sequence file
25 containing the test sequence of inputs for the software application to be tested; and

1 a test monitoring interval field to enable user entry of a timing interval to
2 define how often the test driver program can be monitored to detect a failure of the
3 test driver program.

4
5 **28.** A user interface to enable user entry of parameters to define a model
6 of a software application to be tested, the user interface comprising:

7 an operational modes field to enable user entry of an operational mode of
8 the software application, the operational mode being an attribute of a particular
9 state of the software application; and

10 a modal values field to enable user entry of at least one modal value
11 associated with the operational mode, the modal value describing a behavior of the
12 operational mode.

13
14 **29.** A user interface as recited in claim 28, further comprising an input
15 field to enable user entry of an input of the software application.

16
17 **30.** A user interface as recited in claim 29, further comprising:

18 an operational modes list field to display the operational mode;

19 a modal values list field to display the modal value; and

20 an inputs list field to display the input of the software application.

21
22 **31.** A user interface as recited in claim 28, wherein the user interface
23 enables user initiation of a graph traversal program to generate a test sequence of
24 inputs for the software application.

1 **32.** A user interface as recited in claim 28, wherein the user interface
2 enables user initiation of a test driver program to read a test sequence of inputs for
3 the software application and apply the test sequence to the software application.
4

5 **33.** A user interface to enable user entry of parameters to define a model
6 of a software application to be tested, the user interface comprising:

7 an inputs field to enable user entry of an input of the software application;

8 current state fields to enable user entry of a current state of the software
9 application, the current state being associated with the input; and

10 next state fields to enable user entry of a next state of the software
11 application, the next state indicating the state of the software application after the
12 input has been applied to the current state of the software application.
13

14 **34.** A user interface as recited in claim 33, further comprising a rules
15 field to enable user entry of a rule to describe a transition of the software
16 application from a current state to a next state.
17

18 **35.** A user interface as recited in claim 34, further comprising a control
19 to enable a user to disable a rule such that the model of the software application
20 will be defined without the rule.
21

22 **36.** A user interface as recited in claim 33, wherein:

23 the current state fields include:

24 (1) a current state operational mode field to enable user entry of a
25 current state operational mode of the software application;

(2) a current state modal value field to enable user entry of at least one current state modal value associated with the current state operational mode;

the next state fields include:

(1) a next state operational mode field to enable user entry of a next state operational mode of the software application; and

(2) a next state modal value field to enable user entry of at least one next state modal value associated with the next state operational mode.

37. A user interface as recited in claim 36, wherein:

the current state fields include:

(1) a current state relational operator field to indicate the current state modal value relation to the current state operational mode;

(2) a current state concatenation operator field to indicate the relation between a first current state rule criteria and a second current state rule criteria.

the next state fields include:

(1) a next state relational operator field to indicate the next state modal value relation to the next state operational mode; and

(2) a next state concatenation operator field to indicate the relation between a first next state rule criteria and a second next state rule criteria.

1 **38.** A data structure stored on a computer readable medium comprising:
2 at least one mode data structure to hold an operational mode of a software
3 application to be tested and at least one modal value associated with the
4 operational mode; and
5 at least one rule data structure to hold an input of the software application
6 to be tested, a current state of a software application, and a next state of the
7 software application.

8
9 **39.** A data structure as recited in claim 38, wherein the mode data
10 structure is a linked list of one or more mode data structures.

11
12 **40.** A data structure as recited in claim 38, wherein the rule data
13 structure is a linked list of one or more rule data structures.

14
15 **41.** A data structure as recited in claim 38, wherein:
16 the current state and the next state are defined by rule criteria data
17 structures; and
18 the rule criteria data structures are stored in a linked list.

19
20 **42.** A finite state model-based testing system comprising:
21 a model editor to enable user entry of state information to define a model of
22 a software application to be tested;
23 a rules editor to enable user entry of transition information to further define
24 the model of the software application, the transition information describing a next
25

1 state of the software application after an input has been applied to a current state
2 of the software application;

3 a model generation engine to generate the model of the software
4 application;

5 a graph traversal menu to enable user selection of a graph traversal program
6 to generate a test sequence of inputs for the software application; and

7 a test execution menu to enable user selection of a test driver program to
8 read the test sequence of inputs for the software application and apply the test
9 sequence to the software application.

10
11 **43.** A finite state model-based testing system as recited in claim 42,
12 wherein the model editor comprises:

13 operational modes fields to enable user entry of an operational mode of the
14 software application;

15 modal values fields to enable user entry of a modal value associated with
16 the operational mode; and

17 inputs fields to enable user entry of an input of the software application.

18
19 **44.** A finite state model-based testing system as recited in claim 42,
20 wherein the rules editor comprises fields to display the state information that can
21 be entered using the model editor, the fields comprising:

22 inputs fields to display inputs of the software application, wherein the
23 inputs fields also enable user selection of an input of the software application;

1 operational modes fields to display operational modes of the software
2 application, wherein the operational modes fields also enable user selection of an
3 operational mode; and

4 modal values fields to display modal values associated with an operational
5 mode, wherein the modal values fields also enable user selection of a modal value.
6

7 **45.** A finite state model-based testing system as recited in claim 42,
8 wherein the model is a state table having at least one state table entry, the state
9 table entry having a current state of the software application, an input of the
10 software application, and a next state of the software application; and

11 the model editor enables user initiation of the model generation engine
12 which is configured to evaluate a current state of the software application to
13 determine if an input of the software application can be applied to the current state
14 and in the event that the input can be applied to the current state, writes a state
15 table entry out to the state table.
16

17 **46.** A finite state model-based testing system as recited in claim 42,
18 wherein the model editor facilitates user initiation of the rules editor.
19

20 **47.** A finite state model-based testing system as recited in claim 42,
21 wherein the model editor facilitates user initiation of the graph traversal menu.
22

23 **48.** A finite state model-based testing system as recited in claim 42,
24 wherein the model editor facilitates user initiation of the test execution menu.
25

1 **49.** A computer system comprising:
2 a processor;
3 a memory;
4 a user interface application stored in the memory and executable on the
5 processor to facilitate user definition of a finite-state model to test a software
6 application;
7 the user interface application having computer readable instructions to
8 display a graphical user interface; and
9 a model generation engine stored in memory and executable on the
10 processor to generate the model of the software application.

11
12 **50.** A computer system as recited in claim 49, wherein the user interface
13 enables a user to enter state information and transition information about the
14 software application, the transition information describing a next state of the
15 software application after an input has been applied to a current state of the
16 software application.

17
18 **51.** A computer system as recited in claim 49, wherein the user interface
19 comprises a model editor to enable user entry of operational modes, modal values,
20 and inputs of the software application to define the model.

21
22 **52.** A computer system as recited in claim 49, wherein the user interface
23 comprises a rules editor to enable user entry of an input of the software
24 application, a current state of the software application, and a next state of the
25 software application to define the model.

1
2 **53.** A computer system as recited in claim 49, further comprising at
3 least one graph traversal program stored in the memory and executable on the
4 processor to generate a test sequence of inputs for the software application, the
5 user interface presenting a graph traversal menu to enable a user to select the
6 graph traversal program.

7
8 **54.** A computer system as recited in claim 49, further comprising at
9 least one test driver program stored in the memory and executable on the
10 processor to execute a test sequence of application inputs on the software
11 application, the user interface presenting a test execution menu to enable a user to
12 select the test driver program.

13
14 **55.** A computer system as recited in claim 49, further comprising a data
15 structure stored in the memory, the data structure comprising:

16 at least one mode data structure to hold an operational mode of a software
17 application and at least one modal value associated with the operational mode; and

18 at least one rule data structure to hold an input of the software application, a
19 current state of a software application, and a next state of the software application.
20
21
22
23
24
25

1 **56.** A method comprising:

2 presenting a graphical user interface that facilitates user entry of state
3 information and transition information about a software application to be tested;
4 and

5 generating a model of the software application using the state information
6 and the transition information.

7
8 **57.** A method as recited in claim 56, further comprising:

9 presenting a graphical user interface that facilitates user selection of a graph
10 traversal program; and

11 generating a test sequence of inputs for the software application.

12
13 **58.** A method as recited in claim 56, further comprising:

14 presenting a graphical user interface that facilitates user selection of a test
15 driver program; and

16 executing a test sequence of application inputs on the software application.

17
18 **59.** A method as recited in claim 56, further comprising enabling a user

19 to define a transition rule of the software application, wherein the enabling
20 comprises presenting a graphical user interface to facilitate user entry of an input
21 of the software application, a current state of the software application associated
22 with the input, and a next state of the software application.

1 **60.** A method as recited in claim 56, further comprising:

2 presenting a graphical user interface that facilitates a user defining a
3 transition rule of the software application and disabling the transition rule; and

4 generating the model of the software application without incorporating the
5 disabled transition rule.

6
7 **61.** A method as recited in claim 56, wherein generating a model of the
8 software application comprises:

9 evaluating a current state of the software application to determine if an
10 input of the software application can be applied to the current state; and

11 in the event that the input can be applied to the current state, writing a state
12 table entry out to a state table.

13
14 **62.** A method as recited in claim 56, further comprising enabling a user
15 to define the state information, wherein the enabling comprises presenting a
16 graphical user interface to facilitate user entry of an operational mode of the
17 software application, at least one modal value associated with the operational
18 mode, and an input of the software application.

1 **63.** A method as recited in claim 56, further comprising enabling a user
2 to define the transition information, wherein the enabling comprises presenting a
3 graphical user interface to facilitate user entry of a current state of the software
4 application, the current state being associated with an input of the software
5 application, and a next state of the software application, the next state indicating
6 the state of the software application after the input has been applied to the current
7 state of the software application.

8
9 **64.** A computer-readable medium comprising computer executable
10 instructions that, when executed, direct a computing system to perform the method
11 of claim 56.

12
13 **65.** A method comprising:
14 presenting a user interface that facilitates user entry of state information and
15 transition information about a software application to be tested;
16 initiating, via the user interface, generation of a model of the software
17 application;
18 selecting, via the user interface, a graph traversal program that generates a
19 test sequence of inputs for the software application; and
20 selecting, via the user interface, a test driver program that executes a test
21 sequence of application inputs on the software application.

1 **66.** A computer-readable medium comprising computer executable
2 instructions that, when executed, direct a computing system to perform the method
3 of claim 65.

4
5 **67.** A method comprising:
6 receiving state information about a software application to be tested;
7 receiving transition information about the software application;
8 generating a model of the software application;
9 from the model, generating a test sequence of inputs for the software
10 application with a graph traversal program; and
11 executing a test sequence of application inputs on the software application.

12
13 **68.** A method as recited in claim 67, wherein generating a model of the
14 software application comprises:
15 evaluating a current state of the software application to determine if an
16 input of the software application can be applied to the current state; and
17 in the event that the input can be applied to the current state, writing a state
18 table entry out to a state table.

19
20 **69.** A computer-readable medium comprising computer executable
21 instructions that, when executed, direct a computing system to perform the method
22 of claim 67.

1 **70.** A computer-readable medium comprising computer executable
2 instructions that, when executed, direct a computing system to:
3 receive state information about a software application to be tested;
4 receive transition information about the software application;
5 generate a model of the software application;
6 from the model, generate a test sequence of inputs for the software
7 application with a graph traversal program; and
8 execute a test sequence of application inputs on the software application.

1 **ABSTRACT**

2 A finite state model-based testing system has a user interface to enable a
3 user to enter state information and transition information about a software
4 application to be tested. The user interface further enables a user to initiate a
5 model generation engine to generate a model of the software application from the
6 state information and transition information. A graph traversal menu enables a
7 user to select a graph traversal program to generate a test sequence of inputs for
8 the software application from the model, and a test execution menu enables a user
9 to select a test driver program to read the test sequence of inputs for the software
10 application, and execute the test sequence of inputs on the software application.

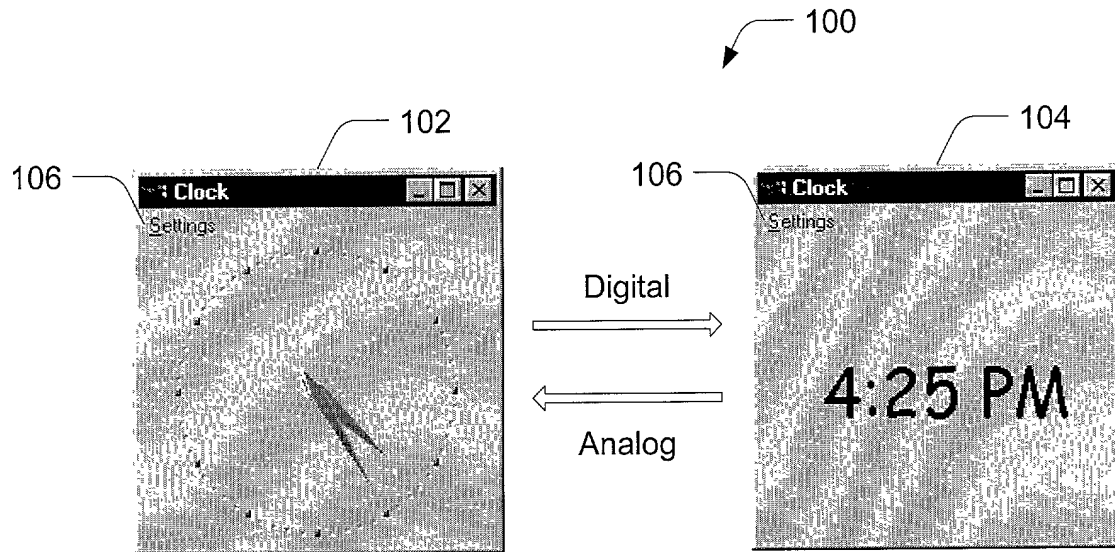


Fig. 1
Prior Art

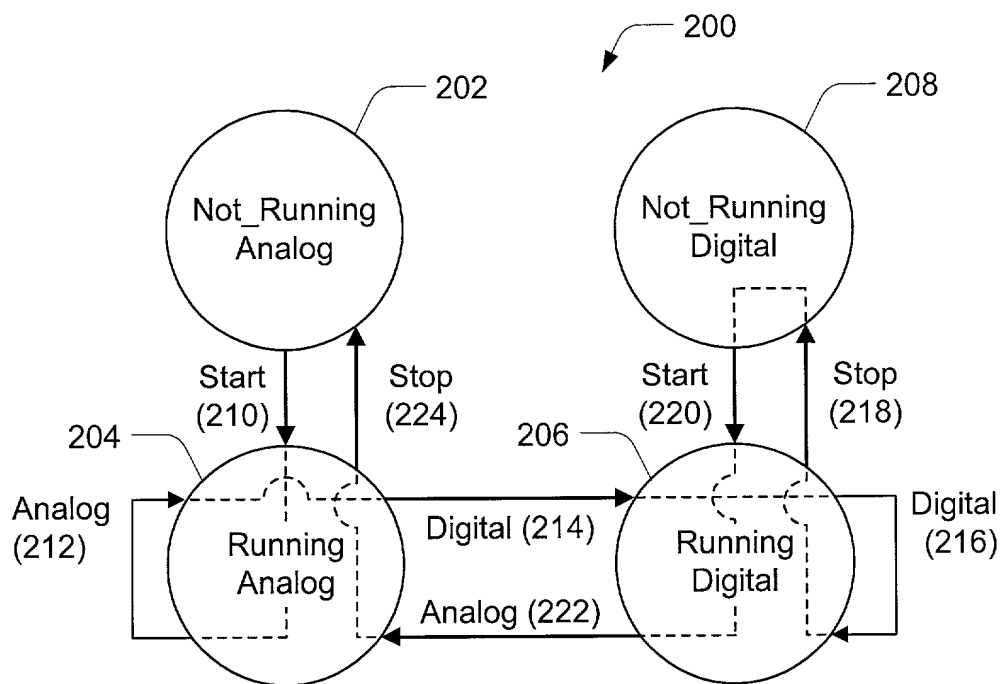
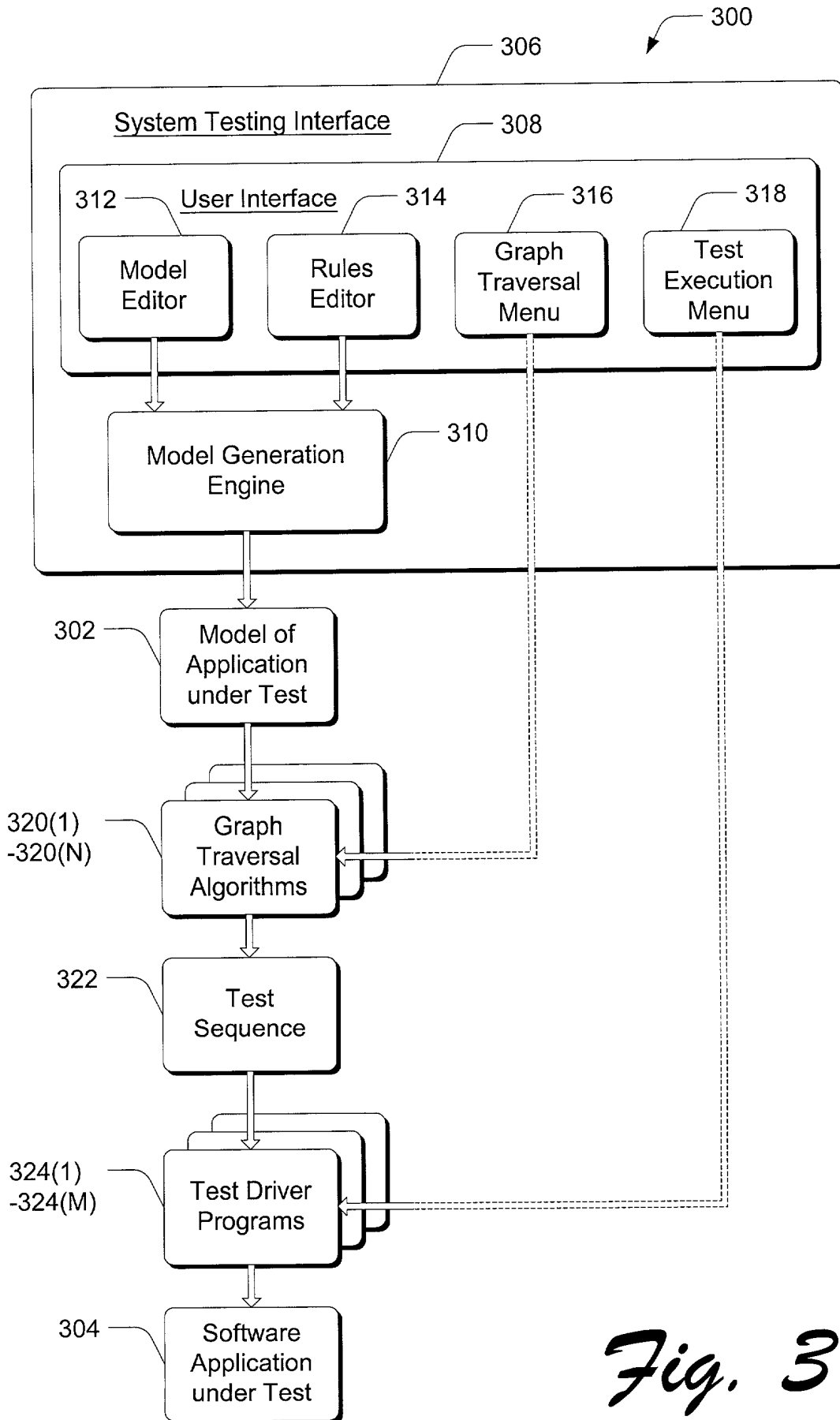


Fig. 2
Prior Art

*Fig. 3*

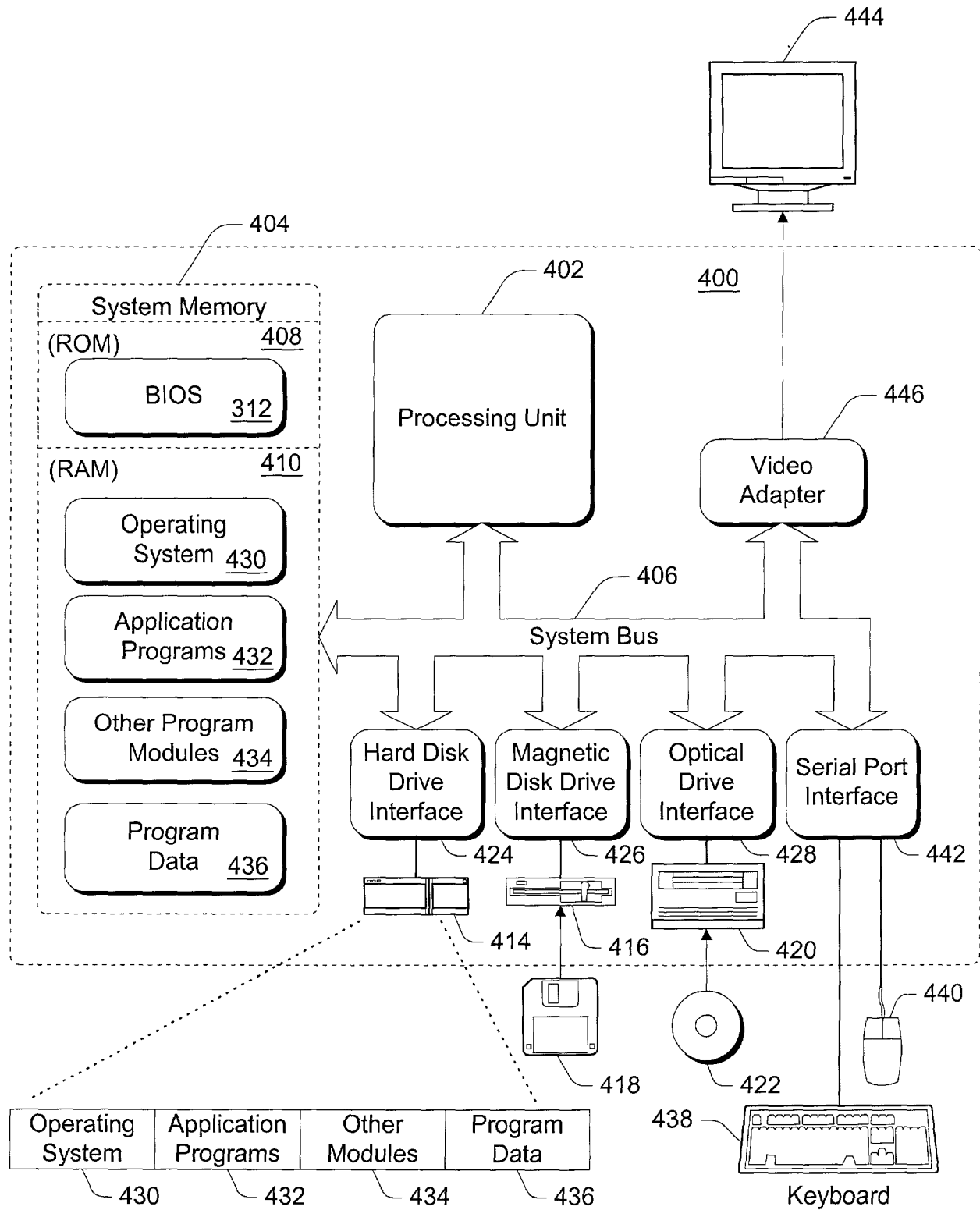
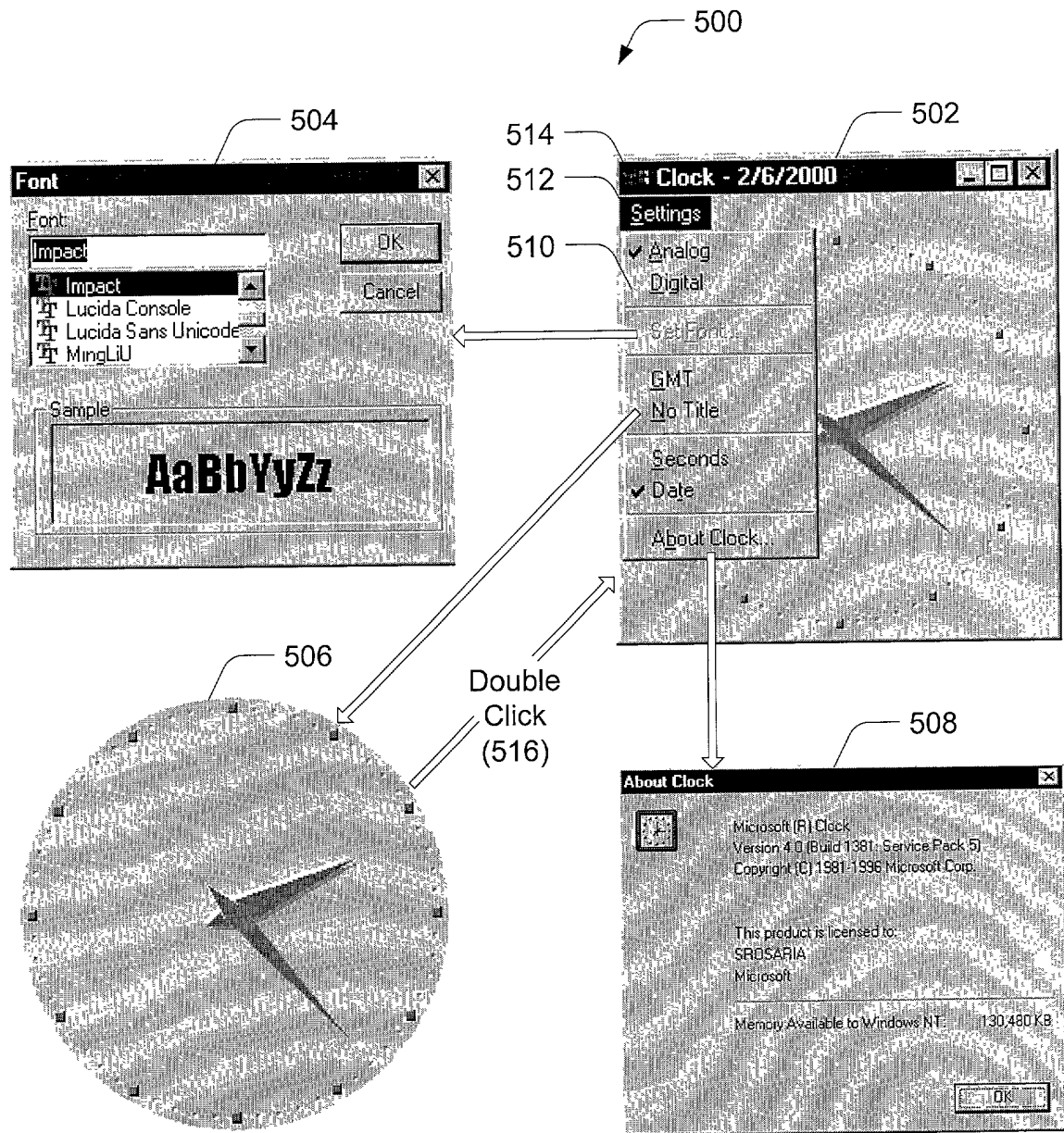
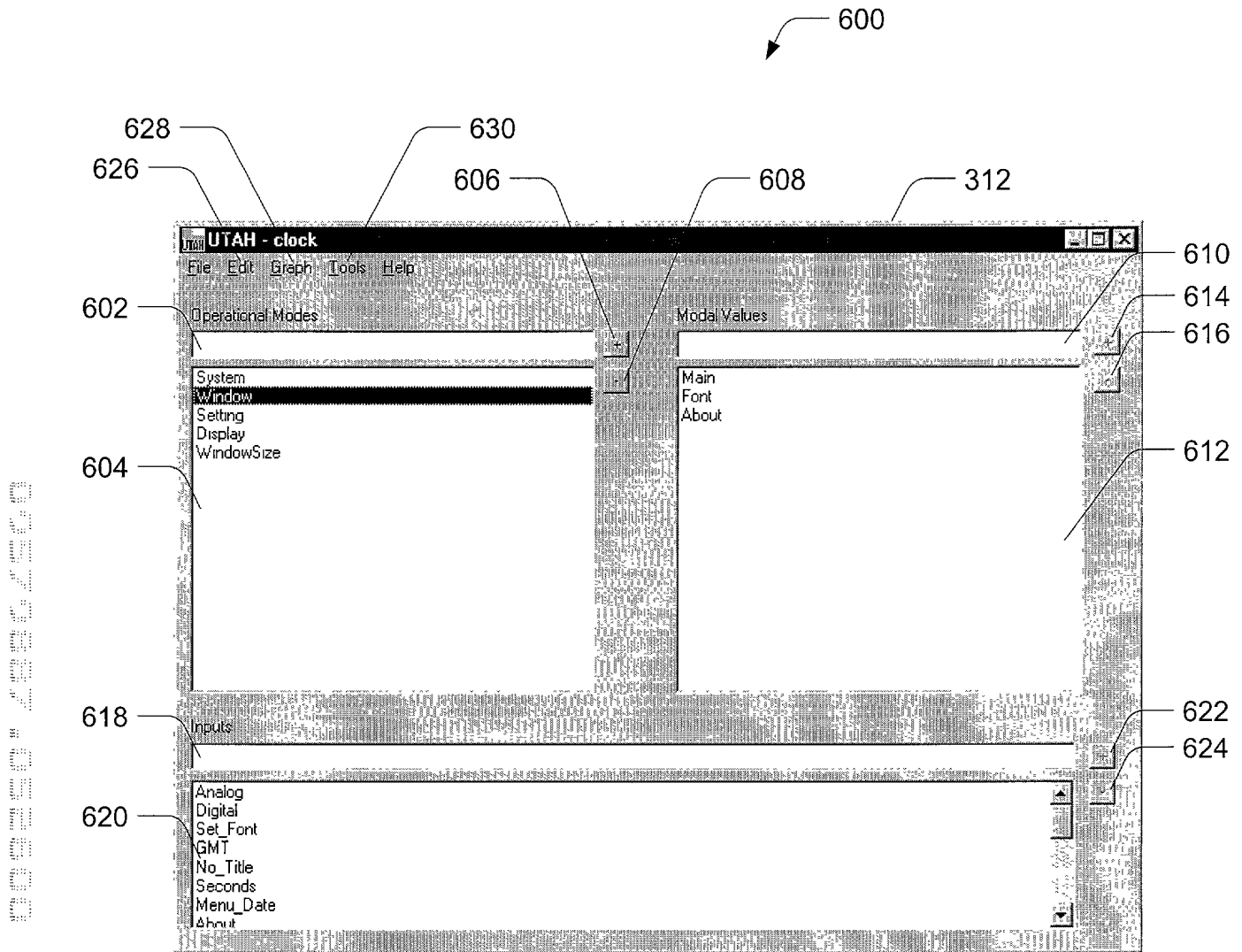


Fig. 4

*Fig. 5*

*Fig. 6*

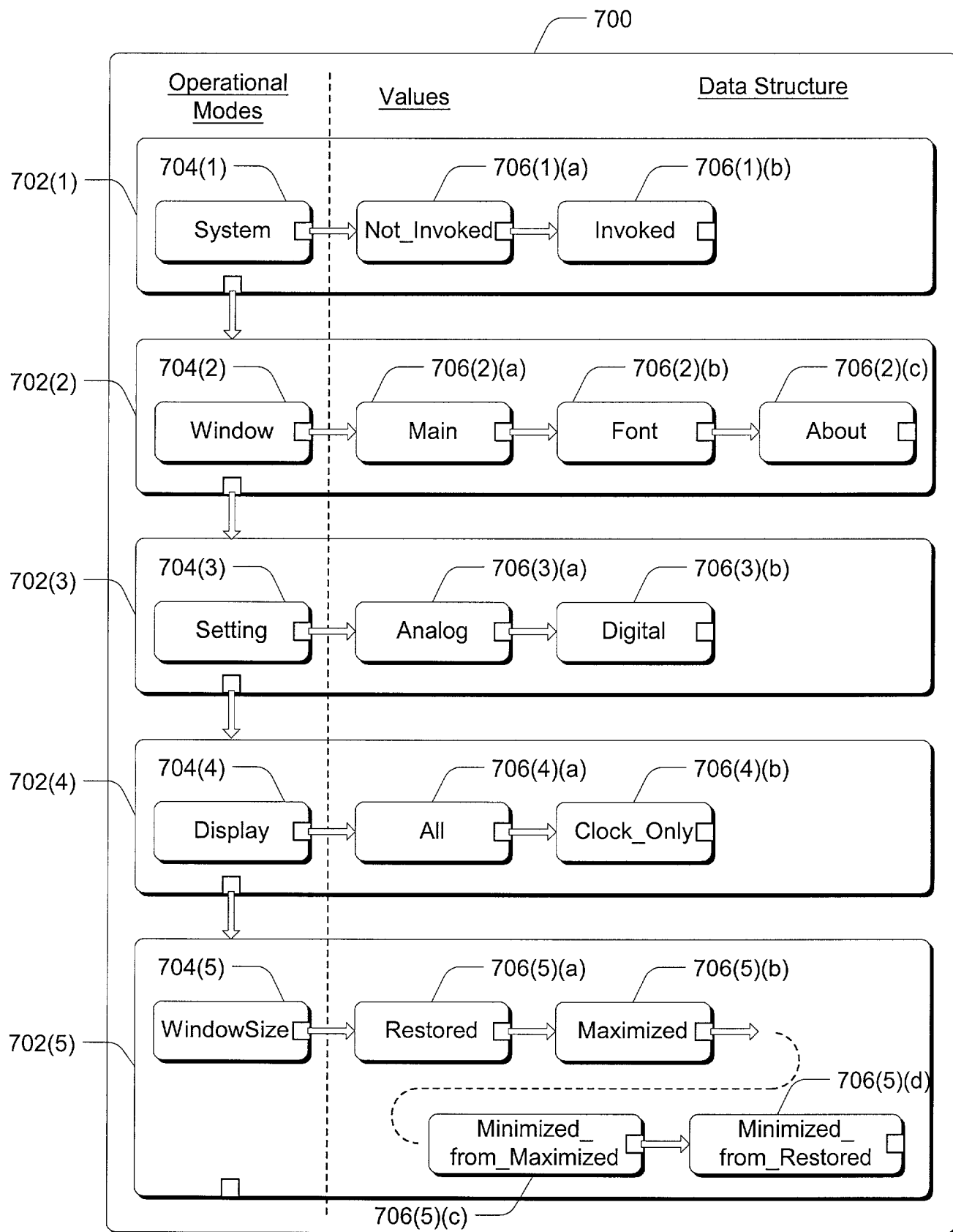


Fig. 7

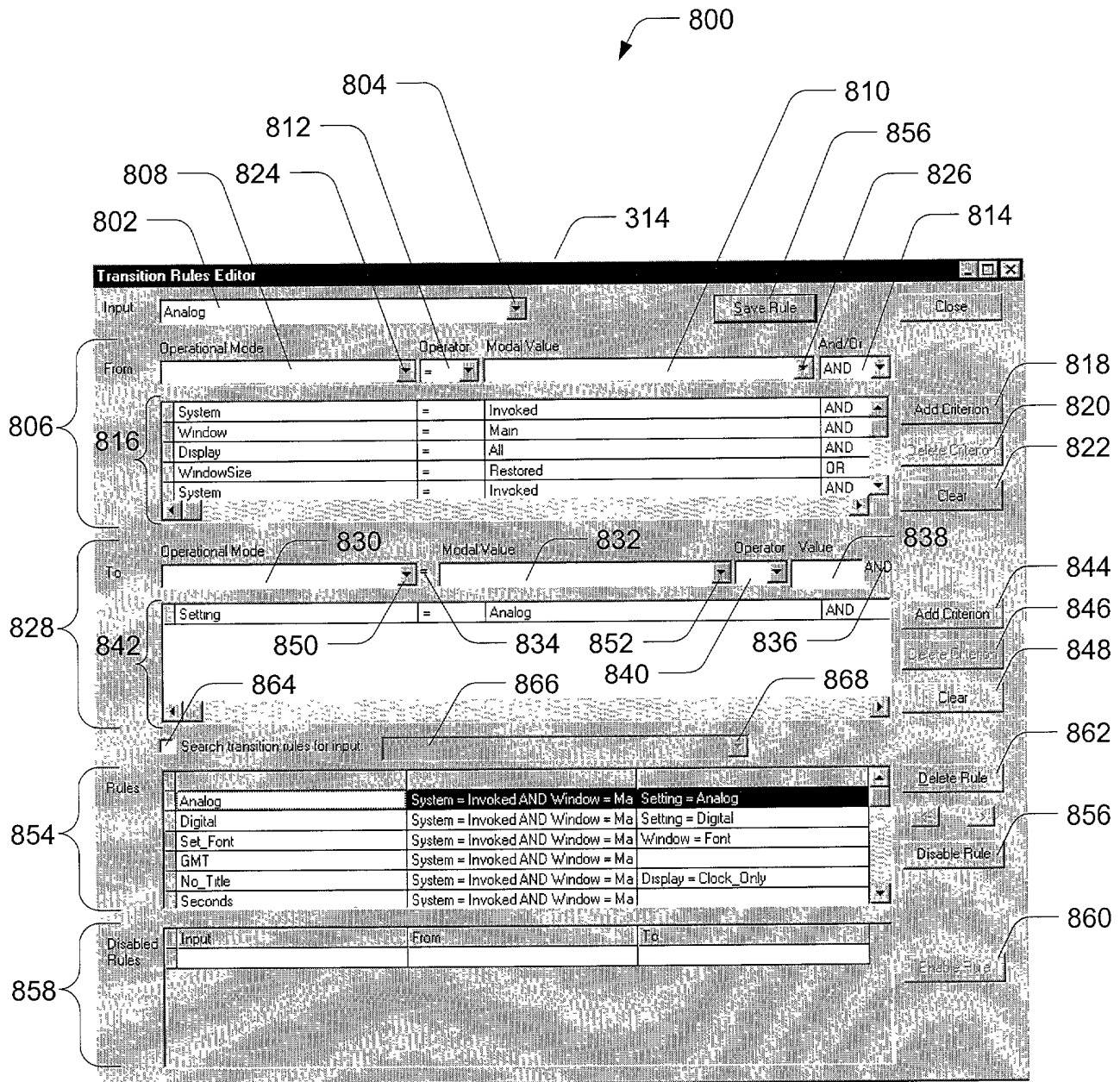


Fig. 8

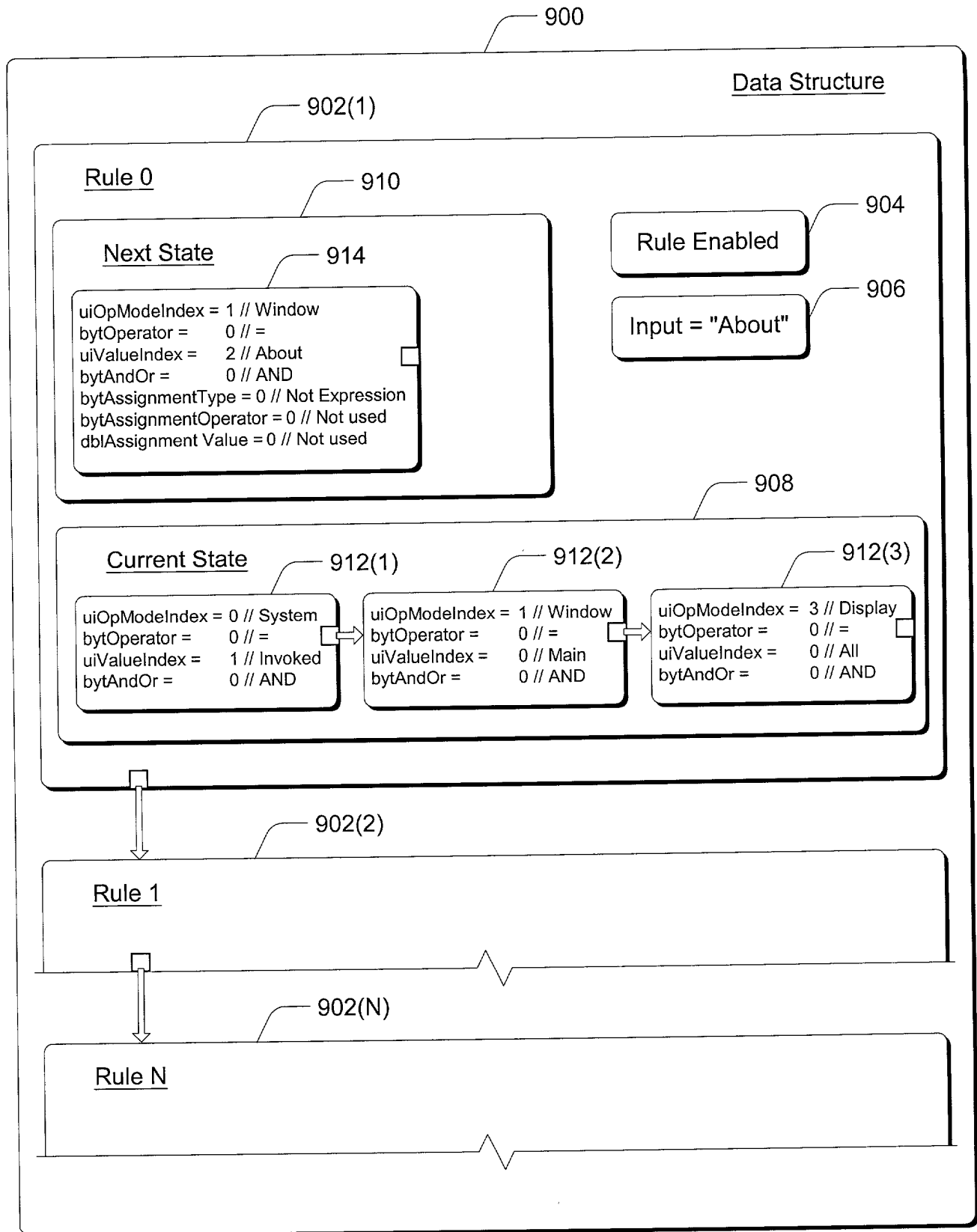
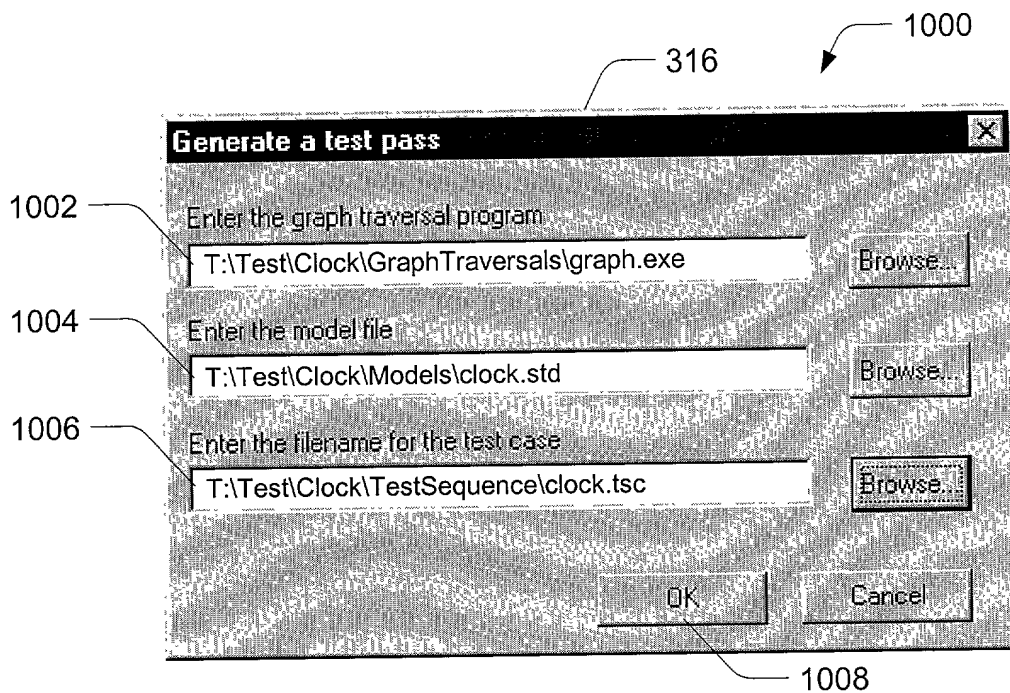
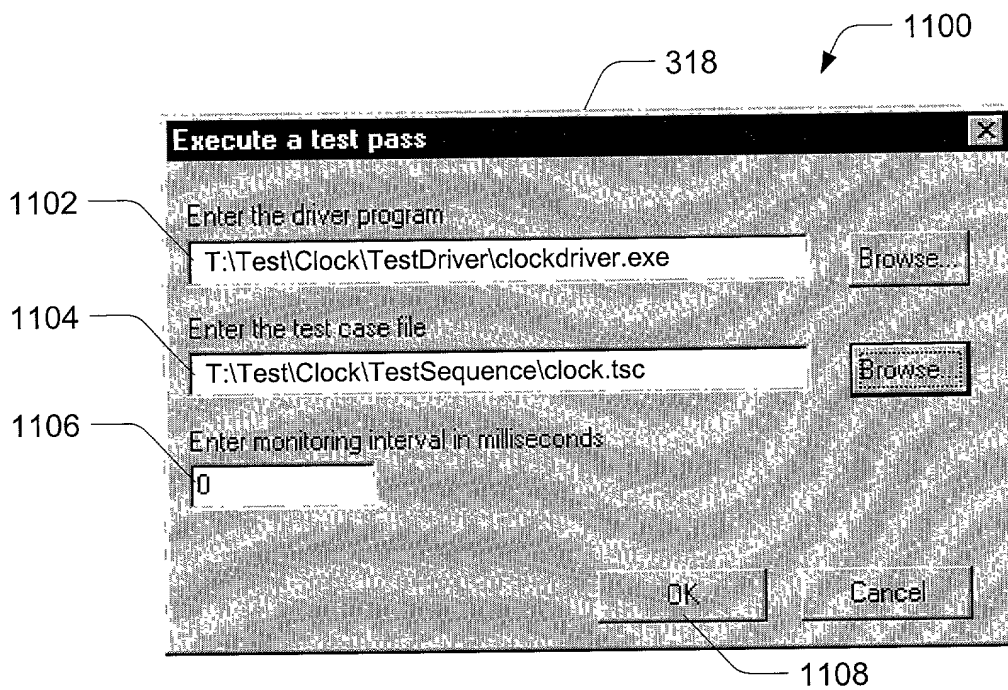
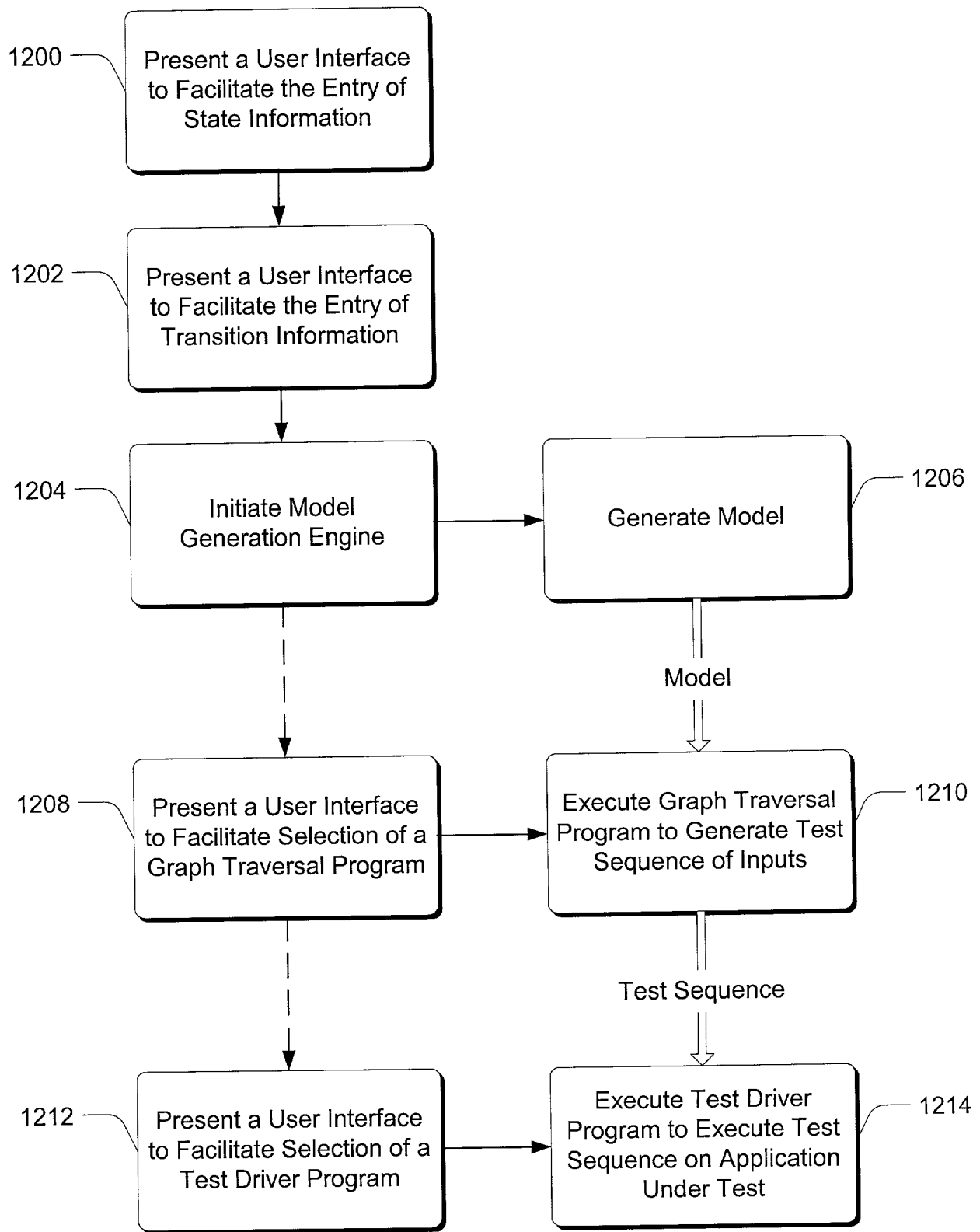


Fig. 9

*Fig. 10**Fig. 11*

*Fig. 12*

1 **IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

2 InventorshipRosaria et al.
3 Applicant Microsoft Corporation
4 Attorney's Docket No. MS1-556US
5 Title: Finite State Model-Based Testing User Interface

6 **DECLARATION FOR PATENT APPLICATION**

7 As a below named inventor, I hereby declare that:

8 My residence, post office address and citizenship are as stated below next to
9 my name.

10 I believe I am the original, first and sole inventor (if only one name is listed
11 below) or an original, first and joint inventor (if plural names are listed below) of the
12 subject matter which is claimed and for which a patent is sought on the invention
13 entitled "Finite State Model-Based Testing User Interface," the specification of
14 which is attached hereto.

15 I have reviewed and understand the content of the above-identified
16 specification, including the claims.

17 I acknowledge the duty to disclose information which is material to the
18 examination of this application in accordance with Title 37, Code of Federal
19 Regulations, § 1.56(a).

20 PRIOR FOREIGN APPLICATIONS: no applications for foreign patents or
21 inventor's certificates have been filed prior to the date of execution of this
22 declaration.

23 **Power of Attorney**

24 I appoint the following attorneys to prosecute this application and transact all
25 future business in the Patent and Trademark Office connected with this application:
26 Lewis C. Lee, Reg. No. 34,656; Daniel L. Hayes, Reg. No. 34,618; Allan T.

1 Sponseller, Reg. 38,318; Steven R. Sponseller, Reg. No. 39,384; James R.
2 Banowsky, Reg. No. 37,773; Lance R. Sadler, Reg. No. 38,605; Michael A. Proksch,
3 Reg. No. 43,021; Thomas A. Jolly, Reg. No. 39,241; David A. Morasch, Reg. No.
4 42,905; Kasey C. Christie, Reg. No. 40,559; Katie E. Sako, Reg. No. 32,628 and
5 Daniel D. Crouse, Reg. No. 32,022.

6 Send correspondence to: LEE & HAYES, PLLC, 421 W. Riverside Avenue,
7 Suite 500, Spokane, Washington, 99201. Direct telephone calls to: David A.
8 Morasch (509) 324-9256.

9
10 All statements made herein of my own knowledge are true and that all
11 statements made on information and belief are believed to be true; and further that
12 these statements were made with the knowledge that willful false statements and the
13 like so made are punishable by fine or imprisonment, or both, under Section 1001 of
14 Title 18 of the United States Code and that such willful false statement may
15 jeopardize the validity of the application or any patent issued therefrom.

16
17 * * * * *

18 Full name of inventor: Steven Rosaria

19 Inventor's Signature



Date: 5/25/2000

20 Residence: Redmond, WA

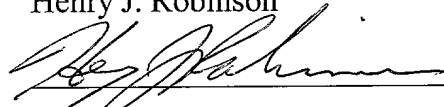
21 Citizenship: Dutch

22 Post Office Address: 7001 Old Redmond Road # M152
23 Redmond, WA 98052
24
25

Full name of inventor:

Henry J. Robinson

Inventor's Signature



Date: 25 MAY 2000

Residence:

Snohomish, WA

Citizenship:

USA

Post Office Address:

20214 108th Drive SE
Snohomish, WA 98296